Package 'units'

January 8, 2018

Version 0.5-1

Title Measurement Units for R Vectors

Depends R (>= 3.0.0)

Imports udunits2 (>= 0.13)

Suggests NISTunits, measurements, xml2, tibble, pillar, knitr, testthat, ggforce, rmarkdown, magrittr

VignetteBuilder knitr

Description Support for measurement units in R vectors, matrices and arrays: automatic propagation, conversion, derivation and simplification of units; raising errors in case of unit incompatibility. Compatible with the POSIXct, Date and difftime classes. Uses the UNIDATA udunits library and unit database for unit compatibility checking and conversion.

License GPL-2

URL https://github.com/r-quantities/units/

BugReports https://github.com/r-quantities/units/issues/

RoxygenNote 6.0.1

Encoding UTF-8

NeedsCompilation no

Author Edzer Pebesma [aut, cre] (0000-0001-8049-7069), Thomas Mailund [aut], Tomasz Kalinowski [aut]

Maintainer Edzer Pebesma <edzer.pebesma@uni-muenster.de>

Repository CRAN

Date/Publication 2018-01-08 14:03:45 UTC

R topics documented:

`as_difftime` *convert units object into difftime object*

Description

convert units object into difftime object

Usage

```
as_difftime(x)
```

Arguments

`x`	object of class `units`

Examples

```
t1 = Sys.time()
t2 = t1 + 3600
d = t2 - t1
du <- as_units(d)
dt = as_difftime(du)
class(dt)
dt
```

as_units *convert object to a units object*

Description

convert object to a units object

difftime objects to units

A number of functions are provided for creating unit objects.

- as_units, a generic with methods for a character string and for quoted language. Note, direct usage of this function by users is typically not necessary, as coercion via as_units is automatically done with `units<-` and set_units().
- make_units(), constructs units from bare expressions. make_units(m/s) is equivalent to as_units(quote(m/s))
- set_units(), a pipe_friendly version of `units<-`. By default it operates with bare expressions like make_unit, but this behavior can be disabled by a specifying mode = "standard" or setting units_options(set_units_mode = "standard").

Usage

```
as_units(x, ...)

## Default S3 method:
as_units(x, value = unitless, ...)

## S3 method for class 'difftime'
as_units(x, value, ...)

make_units(bare_expression, check_is_valid = TRUE)

## S3 method for class 'character'
as_units(x, check_is_valid = TRUE,
  implicit_exponents = NULL, force_single_symbol = FALSE, ...)

## S3 method for class 'call'
as_units(x, check_is_valid = TRUE, ...)
```

Arguments

x	object of class units
...	passed on to other methods
value	an object of class units, or something coercible to one with as_units
bare_expression	a bare R expression describing units. Must be valid R syntax (reserved R syntax words like in must be backticked)

check_is_valid throw an error if all the unit symbols are not either recognized by udunits2 via udunits2::ud.is.parseable(), or a custom user defined via install_symbolic_unit(). If FALSE, no check for validity is performed.

implicit_exponents
If the unit string is in product power form (e.g. "km m-2 s-1"). Defaults to NULL, in which case a guess is made based on the supplied string. Set to TRUE or FALSE if the guess is incorrect.

force_single_symbol
Whether to perform no string parsing and force treatment of the string as a single symbol.

Value

A new unit object that can be used in arithmetic, unit conversion or unit assignment.

Character strings

Generally speaking, there are 3 types of unit strings are accepted in as_units (and by extension, `units<-`).

The first, and likely most common, is a "standard" format unit specification where the relationship between unit symbols or names is specified explicitly with arithmetic symbols for division /, multiplication * and power exponents ^, or other mathematical functions like log(). In this case, the string is parsed as an R expression via parse(text =) after backticking all unit symbols and names, and then passed on to as_units.call(). A heuristic is used to perform backticking, such that any continuous set of characters uninterrupted by one of ()*^- are backticked (unless the character sequence consists solely of numbers 0-9), with some care to not double up on pre-existing backticks. This heuristic appears to be quite robust, and works for units would otherwise not be valid R syntax. For example, percent ("%"), feet ("'"), inches ("in"), and Tesla ("T") are all backticked and parsed correctly.

Nevertheless, for certain complex unit expressions, this backticking heuristic may give incorrect results. If the string supplied fails to parse as an R expression, then the string is treated as a single symbolic unit and symbolic_unit(chr) is used as a fallback with a warning. In that case, automatic unit simplification may not work properly when performing operations on unit objects, but unit conversion and other Math operations should still give correct results so long as the unit string supplied returns TRUE for udunits2::ud.is.parsable().

The second type of unit string accepted is one with implicit exponents. In this format, /, *, and ^, may not be present in the string, and unit symbol or names must be separated by a space. Each unit symbol may optionally be followed by a single number, specifying the power. For example "m2 s-2" is equivalent to "(m^2)*(s^-2)".

The third type of unit string format accepted is the special case of udunits time duration with a reference origin, for example "hours since 1970-01-01 00:00:00". Note, that the handling of time and calendar operations via the udunits library is subtly different from the way R handles date and time operations. This functionality is mostly exported for users that work with udunits time data, e.g., with NetCDF files. Users are otherwise encouraged to use R's date and time functionality provided by Date and POSIXt classes.

Expressions

In as_units(), each of the symbols in the unit expression is treated individually, such that each symbol must be recognized by the udunits database (checked by ud.is.parseable(), *or* be a custom, user-defined unit symbol that was defined either by install_symbolic_unit() or install_conversion_function(). To see which symbols and names are currently recognized by the udunits database, see udunits_symbols().

Note

By default, unit names are automatically substituted with unit names (e.g., kilogram –> kg). To turn off this behavior, set units_options(auto_convert_names_to_symbols = FALSE)

See Also

valid_udunits

Examples

```
s = Sys.time()
d  = s - (s+1)
as_units(d)
# The easiest way to assign units to a numeric vector is like this:
x <- y <- 1:4
units(x) <- "m/s"  # meters / second

# Alternatively, the easiest pipe-friendly way to set units:
if(require(magrittr))
  y %>% set_units(m/s)


# these are different ways of creating the same unit:
# meters per second squared, i.e, acceleration
x1 <- make_units(m/s^2)
x2 <- as_units(quote(m/s^2))
x2 <- as_units("m/s^2")
x3 <- as_units("m s-2") # in product power form, i.e., implicit exponents = T
x4 <- set_units(1,  m/s^2) # by default, mode = "symbols"
x5 <- set_units(1, "m/s^2",   mode = "standard")
x6 <- set_units(1, x1,        mode = "standard")
x7 <- set_units(1, units(x1), mode = "standard")
x8 <- as_units("m") / as_units("s")^2

all_identical <- function(...) {
  l <- list(...)
  for(i in seq_along(l)[-1])
    if(!identical(l[[1]], l[[i]]))
      return(FALSE)
  TRUE
}
all_identical(x1, x2, x3, x4, x5, x6, x7, x8)

# Note, direct usage of these unit creation functions is typically not
```

```
# necessary, since coercion is automatically done via as_units(). Again,
# these are all equivalent ways to generate the same result.

x1 <- x2 <- x3 <- x4 <- x5 <- x6 <- x7 <- x8 <- 1:4
units(x1) <- "m/s^2"
units(x2) <- "m s-2"
units(x3) <- quote(m/s^2)
units(x4) <- make_units(m/s^2)
units(x5) <- as_units(quote(m/s^2))
x6 <- set_units(x6, m/s^2)
x7 <- set_units(x7, "m/s^2", mode = "standard")
x8 <- set_units(x8, units(x1), mode = "standard")

all_identical(x1, x2, x3, x4, x5, x6, x7, x8)


# Both unit names or symbols can be used. By default, unit names are
# automatically converted to unit symbols.
make_units(degree_C)
make_units(kilogram)
make_units(ohm)
# Note, if the printing of non-ascii characters is garbled, then you may
# need to specify the encoding on your system manually like this:
# udunits2::ud.set.encoding("latin1")
# not all unit names get converted to symbols under different encodings

## Arithmetic operations and units
# conversion between unit objects that were defined as symbols and names will
# work correctly, although unit simplification in printing may not always occur.
x <- 500 * make_units(micrograms/liter)
y <- set_units(200, ug/l)
x + y
x * y # numeric result is correct, but units not simplified completely

# note, plural form of unit name accepted too ('liters' vs 'liter'), and
# denominator simplification can be performed correctly
x * set_units(5, liters)

# unit conversion works too
set_units(x, grams/gallon)

## Creating custom, user defined units
# For example, a microbiologist might work with counts of bacterial cells
# make_units(cells/ml) # by default, throws an ERROR
# First define the unit, then the newly defined unit is accepted.
install_symbolic_unit("cells")
make_units(cells/ml)

# Note, install_symbolic_unit() does not add any support for unit
# conversion, or arithmetic operations that require unit conversion. See
# ?install_conversion_function for how to define relationships for user
# defined units.
```

```
## set_units()
# set_units is a pipe friendly version of `units<-`.
if (require(magrittr)) {
 1:5 %>% set_units(N/m^2)
 # first sets to m, then converts to km
 1:5 %>% set_units(m) %>% set_units(km)
}

# set_units has two modes of operation. By default, it operates with
# bare symbols to define the units.
set_units(1:5, m/s)

# use `mode = "standard"` to use the value of supplied argument, rather than
# the bare symbols of the expression. In this mode, set_units() can be
# thought of as a simple alias for `units<-` that is pipe friendly.
set_units(1:5, "m/s", mode = "standard")
set_units(1:5, make_units(m/s), mode = "standard")

# the mode of set_units() can be controlled via a global option
# units_options(set_units_mode = "standard")

# To remove units use
units(x) <- NULL
# or
drop_units(y)
```

`deparse_unit` *deparse unit to string in product power form (e.g. km m-2 s-1)*

Description

deparse unit to string in product power form (e.g. km m-2 s-1)

Usage

```
deparse_unit(x)

as_cf(x)
```

Arguments

x	object of class units

Details

`as_cf` is deprecated; use `deparse_unit`.

Value

length one character vector

Examples

```
u = as_units("kg m-2 s-1", implicit_exponents = TRUE)
u
deparse_unit(u)
```

`drop_units` *drop units*

Description

drop units

Usage

```
drop_units(x)
```

Arguments

`x`	a units object

Value

the numeric without any units attributes, while preserving other attributes like dimensions or other classes.

Note

Equivalent to `units(x) <- NULL`

`hist.units` *histogram for unit objects*

Description

histogram for unit objects

Usage

```
## S3 method for class 'units'
hist(x, xlab = NULL, main = paste("Histogram of", xname),
  ...)
```

Arguments

x	object of class units, for which we want to plot the histogram
xlab	character; x axis label
main	character; title of histogram
...	parameters passed on to hist.default

Examples

```
units_options(parse = FALSE) # otherwise we break on the funny symbol!
u = set_units(rnorm(100), degree_C)
hist(u)
```

---

install_conversion_constant

Install a function for conversion between user-defined units.

Description

Tells the units package how to convert between units that have a linear relationship, i.e. can be related on the form $y = \alpha x$.

Usage

```
install_conversion_constant(from, to, const)
```

Arguments

from	String for the symbol of the unit being converted from.
to	String for the symbol of the unit being converted to.
const	The constant α in the conversion.

Details

This function handles the very common case where units are related through a linear function, that is, you can convert from one to the other as $y = \alpha x$. Using this function, you specify that you can go from values of type from to values of type to by first multiplying a constant and then adding an offset. The function then automatically installs that conversion and the invers $x = y/\alpha$.

For a more general conversion mechanism, see install_conversion_function.

See Also

install_conversion_function

Examples

```
# one orange is worth two apples
install_conversion_constant("orange", "apple", 2)
apples <- 2 * as_units("apple")
oranges <- 1 * as_units("orange")
apples + oranges
oranges + apples
```

`install_conversion_function` *Install a function for conversion between user-defined units.*

Description

Tells the `units` package how to convert one-way from one unit to another.

Usage

```
install_conversion_function(from, to, f)
```

Arguments

`from`	String for the symbol of the unit being converted from.
`to`	String for the symbol of the unit being converted to.
`f`	A function responsible for conversion.

Details

This is the most general way of specifying conversion between user-defined units. The function installes a one-way conversion from one unit to another through a general function, `f`, that must take one numeric argument and return one numeric argument. When the `units` package tries to convert between units, it will look up `from` and `to` to see if it can find a conversion function. If it can, it will call `f` and consider the value converted from unit `from` to unit `to`.

It is the user's responsibility to install a conversion from `to` back to `from` as well. One-way conversion does not work well with the `units` package, since conversion is done in several places for unit expression simplification and if a unit can only be converted in one direction, this simplification will not work correctly.

For conversion that can be done as a linear function, $y = \alpha x + \beta$, you should instead use the `install_conversion_constant` function. This function will automatically install conversion functions for both directions of unit conversion.

See Also

`install_conversion_constant`

Examples

```
install_symbolic_unit("apple")
install_symbolic_unit("orange")
apples <- 2 * as_units("apple")
oranges <- 3 * as_units("orange")

# one orange is worth two apples
install_conversion_function("orange", "apple", function(x) 2 * x)
install_conversion_function("apple", "orange", function(x) x / 2)
apples + oranges
oranges + apples
```

---

install_symbolic_unit *Define new symbolic units*

Description

Adding a symbolic unit allows it to be used in as_units, make_units and set_units. No installation is performed if the unit is already known by udunits.

Usage

```
install_symbolic_unit(chr, warn = TRUE)

uninstall_symbolic_unit(chr, all = FALSE)
```

Arguments

chr	a length 1 character vector that is the new unit name or symbol.
warn	warns if the supplied unit symbol is already a valid unit symbol recognized by udunits.
all	if TRUE, uninstalls all user defined custom symbolic units

---

make_unit *Deprecated functions*

Description

The following functions are deprecated and will be removed in a future release.

Usage

```
make_unit(chr)

parse_unit(chr)

as.units(x, value = unitless)
```

Arguments

chr	length 1 character string
x	a numeric
value	a units object, by default, unitless

Math.units *Mathematical operations for units objects*

Description

Mathematical operations for units objects

Usage

```
## S3 method for class 'units'
Math(x, ...)
```

Arguments

x	object of class units
...	parameters passed on to the Math functions

Examples

```
a <- sqrt(1:10)
a <- set_units(a, m/s)
log(a)
log(a, base = 10)
cumsum(a)
signif(a, 2)
```

matmult *matrix multiplication*

Description

matrix multiplication

Usage

```
x %*% y

## Default S3 method:
x %*% y

## S3 method for class 'units'
x %*% y
```

Arguments

x	numeric matrix or vector
y	numeric matrix or vector

Details

see "%*%" for the base function, reimplemented as default method

Examples

```
a = set_units(1:5, m)
a %*% a
a %*% t(a)
a %*% set_units(1:5, 1)
set_units(1:5, 1) %*% a
```

Ops.units *S3 Ops Group Generic Functions for units objects*

Description

Ops functions for units objects, including comparison, product and divide, add, subtract

Usage

```
## S3 method for class 'units'
Ops(e1, e2)
```

Arguments

e1	object of class units, or something that can be coerced to it by as_units(e1)
e2	object of class units, or something that can be coerced to it by as_units(e2), or in case of power a number (integer n or 1/n)

Value

object of class units

Examples

```
a <- set_units(1:3, m/s)
b <- set_units(1:3, m/s)
a + b
a * b
a / b
a <- as_units("kg m-3")
b <- set_units(1, kg/m/m/m)
a + b
a = set_units(1:5, m)
a %/% a
a %/% set_units(2)
set_units(1:5, m^2) %/% set_units(2, m)
a %% a
a %% set_units(2 )
```

---

plot.units *create axis label with appropriate labels*

Description

create axis label with appropriate labels

plot unit objects

Usage

```
make_unit_label(lab, u, sep = units_options("sep"),
  group = units_options("group"), parse = units_options("parse"))

## S3 method for class 'units'
plot(x, y, xlab = NULL, ylab = NULL, ...)
```

Arguments

lab	length one character; name of the variable to plot
u	vector of class units

sep	length two character vector, defaulting to c("~","~"), with the white space between unit name and unit symbols, and between subsequent symbols.
group	length two character vector with grouping symbols, e.g. c("(",")") for parenthesis, or c("","") for no group symbols
parse	logical; indicates whether a parseable expression should be returned (typically needed for super scripts), or a simple character string without special formatting.
x	object of class units, to plot along the x axis, or, if y is missing, along the y axis
y	object to plot along the y axis, or missing
xlab	character; x axis label
ylab	character; y axis label
...	other parameters, passed on to plot.default

Details

units_options can be used to set and change the defaults for sep, group and doParse.

Examples

```
oldpar = par(mar = par("mar") + c(0, .3, 0, 0))
displacement = mtcars$disp * ud_units[["in"]]^3
# an example that would break if parse were (default) TRUE, since 'in' is a reserved word:
units_options(parse=FALSE)
make_unit_label("displacement", displacement)
units_options(parse=TRUE)
units(displacement) = with(ud_units, cm^3)
weight = mtcars$wt * 1000 * with(ud_units, lb)
units(weight) = with(ud_units, kg)
plot(weight, displacement)
units_options(group = c("(", ")") )  # parenthesis instead of square brackets
plot(weight, displacement)
units_options(sep = c("~~~", "~"), group = c("", ""))  # no brackets; extra space
plot(weight, displacement)
units_options(sep = c("~", "~~"), group = c("[", "]"))
gallon = as_units("gallon")
consumption = mtcars$mpg * with(ud_units, mi/gallon)
units(consumption) = with(ud_units, km/l)
plot(displacement, consumption) # division in consumption
units_options(negative_power = TRUE) # division becomes ^-1
plot(displacement, consumption)
plot(1/displacement, 1/consumption)
par(oldpar)
```

seq.units *seq method for units objects*

Description

seq method for units objects

Usage

```
## S3 method for class 'units'
seq(from, to, by = ((to - from)/(length.out - 1)),
  length.out = NULL, along.with = NULL, ...)
```

Arguments

from	see seq
to	see seq
by	see seq
length.out	see seq
along.with	see seq
...	see seq

Details

arguments with units are converted to have units of the first argument (which is either from or to)

Examples

```
seq(to = set_units(10, m), by = set_units(1, m), length.out = 5)
seq(set_units(10, m), by = set_units(1, m), length.out = 5)
seq(set_units(10, m), set_units(19, m))
seq(set_units(10, m), set_units(.1, km), set_units(10000, mm))
```

set_units *set_units*

Description

A pipe friendly version of units<-

Usage

```
set_units(x, value, ..., mode = units_options("set_units_mode"))
```

Arguments

x	a numeric to be assigned units, or a units object to have units converted
value	a units object, or something coercible to one with as_units. Depending on mode, the unit is constructed from the supplied bare expression or from the supplied value via standard evaluation.
...	passed on to as_units
mode	if "symbols" (the default), then unit is constructed from the expression supplied. Otherwise, ifmode = "standard", standard evaluation is used for the supplied value This argument can be set via a global option units_options(set_units_mode = "standard")

See Also

as_units

---

tibble *type_sum function for units*

Description

type_sum function for units

pillar_shaft function for units

Usage

```
type_sum.units(x, ...)

pillar_shaft.units(x, ...)
```

Arguments

x	see type_sum
...	see type_sum

---

ud_units *List containing pre-defined units from the udunits2 package.*

Description

Lazy loaded when used

Usage

```
ud_units
```

Format

An object of class NULL of length 0.

unitless *The "unit" type for vectors that are actually dimension-less.*

Description

The "unit" type for vectors that are actually dimension-less.

Usage

```
unitless
```

Format

An object of class `symbolic_units` of length 2.

units *Set measurement units on a numeric vector*

Description

Set measurement units on a numeric vector

Convert units

retrieve measurement units from `units` object

Usage

```
## S3 replacement method for class 'numeric'
units(x) <- value

## S3 replacement method for class 'units'
units(x) <- value

## S3 replacement method for class 'logical'
units(x) <- value

## S3 method for class 'units'
units(x)
```

Arguments

`x`	numeric vector, or object of class `units`
`value`	object of class `units` or `symbolic_units`, or in the case of `set_units` expression with symbols that can be resolved in ud_units (see examples).

Value

object of class `units`

the units method retrieves the units attribute, which is of class `symbolic_units`

Examples

```
x = 1:3
class(x)
units(x) <- with(ud_units, m/s) # valid
class(x)
y = 2:5
a <- with(ud_units, 1:3 * m/s)
units(a) <- with(ud_units, km/h)
a
```

---

units_options *set one or more units global options*

Description

set units global options, mostly related how units are printed and plotted

Usage

```
units_options(..., sep, group, negative_power, parse, set_units_mode,
  auto_convert_names_to_symbols)
```

Arguments

`...`	ignored
`sep`	character length two; default `c("~", "~")`; space separator between variable and units, and space separator between two different units
`group`	character length two; start and end group, may be two empty strings, a parenthesis pair, or square brackets.
`negative_power`	logical, default FALSE; should denominators have negative power, or follow a division symbol?
`parse`	logical, default `TRUE`; should the units be made into an expression (so we get subscripts)? Setting to `FALSE` may be useful if parse fails, e.g. if the unit contains symbols that assume a particular encoding
`set_units_mode`	character; either `"symbols"` or `"standard"`; see set_units
`auto_convert_names_to_symbols`	logical: should names, such as `degree_C` be converted to their usual symbol?

Examples

```
units_options(sep = c("~~~", "~"), group = c("", "")) # more space, parenthesis
## set defaults:
units_options(sep = c("~", "~"), group = c("[", "]"), negative_power = FALSE, parse = TRUE)
```

valid_udunits *Get information about valid units*

Description

The returned dataframe is constructed at runtime by reading the xml database that powers unit conversion in [package:udunits2]. Inspect this dataframe to determine what inputs are accepted by `as_units` (and the other functions it powers: `make_units` , `set_units` , `units<-`).

Usage

```
valid_udunits(quiet = FALSE)

valid_udunits_prefixes(quiet = FALSE)
```

Arguments

`quiet`	logical, defaults `TRUE` to give a message about the location of the udunits database being read.

Details

Any entry listed under `symbol` , `symbol_aliases` , `name_singular` , `name_singular_aliases` , `name_plural` , or `name_plural_aliases` is valid. Additionally, any entry under `symbol` or `symbol_aliases` may can also contain a valid prefix, as specified by `valid_udunits_prefixes()` .

Note, this is primarily intended for interactive use, the exact format of the returned dataframe may change in the future.

Value

a data frame with columns `symbol` , `symbol_aliases` , `name_singular` , `name_singular_aliases` , `name_plural` , or `name_plural_aliases` , `def` , `definition` , `comment` , `dimensionless` and `source_xml`

Examples

```
valid_udunits()
valid_udunits_prefixes()
if(interactive())
  View(valid_udunits())
```

Index