Package ‘units’

June 13, 2020

Version 0.6-7

Title Measurement Units for R Vectors

Depends R (>= 3.0.2)

Imports Rcpp

LinkingTo Rcpp (>= 0.12.10)

Suggests udunits2, NISTunits, measurements, xml2, magrittr, pillar (>=
1.3.0), dplyr (>= 1.0.0), vctrs (>= 0.3.1), knitr, testthat,
ggforce, rmarkdown

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.
Documentation about 'units' is provided in the paper by Pebesma, Mailund &
Hiebert (2016, <doi:10.32614/RJ-2016-061>), included in this package as a
vignette; see 'citation(``units'')' for details.

SystemRequirements udunits-2

License GPL-2

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

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

RoxygenNote 7.1.0

Encoding UTF-8

NeedsCompilation yes

Author Edzer Pebesma [aut, cre] (<https://orcid.org/0000-0001-8049-7069>),
Thomas Mailund [aut],
Tomasz Kalinowski [aut],
James Hiebert [ctb],
Iñaki Ucar [ctb] (<https://orcid.org/0000-0001-6403-5550>)

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

Repository CRAN

Date/Publication 2020-06-13 15:30:02 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

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 `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 `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_constant()`. 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(requireNamespace("magrittr", quietly = TRUE)) {
  library(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:
# 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_constant for defining relationships between user
# defined units.

## set_units()
# set_units is a pipe friendly version of `units<-`.
if(requireNamespace("magrittr", quietly = TRUE)) {
  library(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
set_units(x, NULL)
# or
drop_units(y)
```

`boxplot.units` *boxplot for unit objects*

Description

boxplot for unit objects

Usage

```
## S3 method for class 'units'
boxplot(x, ..., horizontal = FALSE)
```

Arguments

`x`	object of class units, for which we want to plot the boxplot
`...`	parameters passed on to boxplot.default
`horizontal`	logical indicating if the boxplots should be horizontal; default FALSE means vertical boxes.

Examples

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

`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 attribute and class.

Usage

```
drop_units(x)

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

## S3 method for class 'data.frame'
drop_units(x)

## S3 method for class 'mixed_units'
drop_units(x)
```

Arguments

`x`	an object with units metadata.

Details

Equivalent to `units(x) <-NULL`, or the pipe-friendly version `set_units(x,NULL)`, but `drop_units` will fail if the object has no units metadata. Use the alternatives if you want this operation to succeed regardless of the object type.

A `data.frame` method is also provided, which checks every column and drops units if any.

Value

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

Examples

```
x <- 1
y <- set_units(x, m/s)

# this succeeds
drop_units(y)
set_units(y, NULL)
set_units(x, NULL)

## Not run:
# this fails
```

```
drop_units(x)

## End(Not run)

df <- data.frame(x=x, y=y)
df
drop_units(df)
```

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 conversion constant or offset 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 $y = \alpha x$ (constant) or $y = \alpha + x$ (offset).

Usage

```
install_conversion_constant(from, to, const)

install_conversion_offset(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. One of `from` and `to` must be an existing unit name.
`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 multiplying by a constant, or adding a constant.

See Also

`install_symbolic_unit`, `remove_symbolic_unit`

Examples

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

install_conversion_offset("meter", "newmeter", 1)
m = set_units(1:3, meter)
n = set_units(1:3, newmeter)
m + n
n + m
```

`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(name, warn = TRUE, dimensionless = TRUE)

remove_symbolic_unit(name)
```

Arguments

`name`	a length 1 character vector that is the unit name or symbol.
`warn`	warns if the supplied unit symbol is already a valid unit symbol recognized by udunits.
`dimensionless`	logical; if `TRUE`, a new dimensionless unit is created, if `FALSE` a new base unit is created. Dimensionless units are convertible to other dimensionless units (such as `rad`), new base units are not convertible to other existing units.

Details

`install_symbolic_unit` installs a new dimensionless unit; these are directly compatible to any other dimensionless unit. To install a new unit that is a scaled or shifted version of an existing unit, use `install_conversion_constant` or `install_conversion_offset` directly.

See Also

`install_conversion_constant`, `install_conversion_offset`

Examples

```
install_symbolic_unit("person")
set_units(1, rad) + set_units(1, person) # that is how dimensionless units work!
```

`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

Details

Logarithms receive a special treatment by the underlying **udunits2** library. If a natural logarithm is applied to some `unit`, the result is `ln(re 1 unit)`, which means *natural logarithm referenced to* `1 unit`. For base 2 and base 10 logarithms, the output `lb(...)` and `lg(...)` respectively instead of `ln(...)`.

This is particularly important for some units that are typically expressed in a logarithmic scale (i.e., *bels*, or, more commonly, *decibels*), such as Watts or Volts. For some of these units, the default **udunits2** database contains aliases: e.g., `BW` (bel-Watts) is an alias of `lg(re 1 W)`; `Bm` (bel-milliWatts) is an alias of `lg(re 0.001 W)`; `BV` is an alias of `lg(re 1 V)` (bel-Volts), and so on and so forth (see the output of `valid_udunits()` for further reference).

Additionally, the **units** package defines `B`, the *bel*, by default (because it is not defined by **udunits2**) as an alias of `lg(re 1)`, unless a user-provided XML database already contains a definition of `B`, or the `define_bel` option is set to `FALSE` (see `help(units_options)`).

Examples

```
# roundings, cummulative functions
x <- set_units(sqrt(1:10), m/s)
signif(x, 2)
cumsum(x)

# trigonometry
sin(x) # not meaningful
```

```
x <- set_units(sqrt(1:10), rad)
sin(x)
cos(x)
x <- set_units(seq(0, 1, 0.1), 1)
asin(x)
acos(x)

# logarithms
x <- set_units(sqrt(1:10), W)
log(x) # base exp(1)
log(x, base = 3)
log2(x)
log10(x)
set_units(x, dBW) # decibel-watts
set_units(x, dBm) # decibel-milliwatts
```

`mixed_units` *Create or convert to a mixed units list-column*

Description

Create or convert to a mixed units list-column

Usage

```
mixed_units(x, values, ...)

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

Arguments

`x`	numeric, or vector of class `units`
`values`	character vector with units encodings, or list with symbolic units of class `mixed_symbolic_units`
`...`	ignored
`value`	see values

Details

if `x` is of class `units`, `values` should be missing or of class `mixed_symbolic_units`; if `x` is numeric, `values` should be a character vector the length of `x`.

Examples

```
a <- 1:4
u <- c("m/s", "km/h", "mg/L", "g")
mixed_units(a, u)
units(a) = as_units("m/s")
mixed_units(a) # converts to mixed representation
```

`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

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).

Details

if `value` is of class `units` and has a value unequal to 1, this value is ignored unless `units_options("simplifiy")` is `TRUE`. If `simplify` is `TRUE`, `x` is multiplied by this value.

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) <- as_units("m/s")
class(x)
y = 2:5
a <- set_units(1:3, m/s)
units(a) <- with(ud_units, km/h)
a
# convert to a mixed_units object:
units(a) = c("m/s", "km/h", "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, simplify, allow_mixed, unitless_symbol,
  define_bel)
```

Arguments

`...`	named options (character) for which the value is queried
`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; default: 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; default is `"symbols"`
`auto_convert_names_to_symbols`	logical, default `TRUE`: should names, such as `degree_C` be converted to their usual symbol?
`simplify`	logical, default `NA`; simplify units in expressions?
`allow_mixed`	logical; if `TRUE`, combining mixed units creates a `mixed_units` object, if `FALSE` it generates an error
`unitless_symbol`	character; set the symbol to use for unitless (1) units
`define_bel`	logical; if `TRUE`, define the unit `B` (i.e., the *bel*, widely used with the *deci-* prefix as `dB`, *decibel*) as an alias of `lg(re 1)`. `TRUE` by default, unless `B` is already defined in the existing XML database.

Details

This sets or gets units options. Set them by using named arguments, get them by passing the option name.

The default `NA` value for `simplify` means units are not simplified in set_units or as_units, but are simplified in arithmetical expressions.

Value

in case options are set, invisibly a named list with the option values that are being set; if an option is queried, the current option value.

Examples

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

`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: `as_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

```
if (requireNamespace("xml2", quietly = TRUE)) {
  valid_udunits()
  valid_udunits_prefixes()
  if(interactive())
    View(valid_udunits())
}
```

Index