Package ‘photobiology’

August 1, 2025

Type Package

Title Photobiological Calculations

Version 0.13.2

Date 2025-07-31

Maintainer Pedro J. Aphalo <pedro.aphalo@helsinki.fi>

Description Definitions of classes, methods, operators and functions for use in photobiology and radiation meteorology and climatology. Calculation of effective (weighted) and not-weighted irradiances/doses, fluence rates, transmittance, reflectance, absorptance, absorbance and diverse ratios and other derived quantities from spectral data. Local maxima and minima: peaks, valleys and spikes. Conversion between energy-and photon-based units. Wavelength interpolation. Colours and vision. This package is part of the 'r4photobiology' suite, Aphalo, P. J. (2015) <doi:10.19232/uv4pb.2015.1.14>.

License GPL (>= 2)

Depends R (>= 4.1.0), SunCalcMeeus (>= 0.1.2)

Imports stats, grDevices, polynom (>= 1.4-1), tibble (>= 3.2.0), stringr (>= 1.4.0), lubridate (>= 1.9.3), caTools (>= 1.18.0), plyr (>= 1.8.9), dplyr (>= 1.1.4), tidyr (>= 1.3.1), splus2R (>= 1.3-3), zoo (>= 1.8-12), rlang (>= 1.1.4)

Suggests knitr (>= 1.48), rmarkdown (>= 2.27), testthat (>= 3.2.1), roxygen2 (>= 7.3.2), lutz (>= 0.3.2), covr (>= 3.6.0)

LazyLoad yes

LazyData yes

ByteCompile true

URL `https://docs.r4photobiology.info/photobiology/`, `https://github.com/aphalo/photobiology`

BugReports `https://github.com/aphalo/photobiology/issues`

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3385-972X>),
Titta K. Kotilainen [ctb] (ORCID:
<https://orcid.org/0000-0002-2822-9734>),
Glenn Davis [ctb],
Agnese Fazio [ctb]

Repository CRAN

Date/Publication 2025-07-31 23:50:02 UTC

Contents

`photobiology-package` *photobiology: Photobiological Calculations*

Description

Definitions of classes, methods, operators and functions for use in photobiology and radiation meteorology and climatology. Calculation of effective (weighted) and not-weighted irradiances/doses, fluence rates, transmittance, reflectance, absorptance, absorbance and diverse ratios and other derived quantities from spectral data. Local maxima and minima: peaks, valleys and spikes. Conversion between energy-and photon-based units. Wavelength interpolation. Colours and vision. This package is part of the 'r4photobiology' suite, Aphalo, P. J. (2015) doi:10.19232/uv4pb.2015.1.14.

Details

Package 'photobiology' is at the core of a suite of R packages supporting computations and plotting relevant to photobiology (described at `https://www.r4photobiology.info/`). Package 'photobiology' has its main focus in the characterization of the light environment, the description of optical properties of objects and substances and description of light responses of organisms and devices used to measure light. The facilities for spectral data storage and manipulations are widely useful in photobiology, chemistry, geophysics, radiation climatology and remote sensing. Astronomical computations for the sun are also implemented. The design of object classes for spectral data supports reproducibility by facilitating the consistent use of units and physical quantities and consistent embedding of metadata. Data are expressed throughout using SI base units, except for wavelengths which are consistently expressed in nanometres $[nm]$. Please see the vignette *0: The R for photobiology Suite* for a description of the suite.

Acknowledgements

This work was funded by the Academy of Finland (decision 252548). COST Action FA9604 'UV4Growth' facilitated discussions and exchanges of ideas that lead to the development of this package. The contributions of Andy McLeod, Lars Olof Björn, Nigel Paul, Lasse Ylianttila, T. Matthew Robson and Titta Kotilainen were specially significant. Tutorials by Hadley Wickham and comments on my presentation at UseR!2015 allowed me to significantly improve the coding and functionality.

Author(s)

Maintainer: Pedro J. Aphalo <`pedro.aphalo@helsinki.fi`> (ORCID)

Other contributors:

- Titta K. Kotilainen (ORCID) [contributor]
- Glenn Davis <`gdavis@gluonics.com`> [contributor]
- Agnese Fazio <`agnese.fazio@uni-jena.de`> [contributor]

References

Aphalo, P. J., Albert, A., Björn, L. O., McLeod, A. R., Robson, T. M., Rosenqvist, E. (Eds.). (2012). *Beyond the Visible: A handbook of best practice in plant UV photobiology* (1st ed., p. xx + 174). Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). Open access PDF download available at doi:10.31885/9789521083631.

Aphalo, Pedro J. (2015) The r4photobiology suite. *UV4Plants Bulletin*, 2015:1, 21-29. doi:10.19232/uv4pb.2015.1.14.

Maia, R., Eliason, C. M., Bitton, P. P., Doucet, S. M., Shawkey, M. D. (2013) pavo: an R package for the analysis, visualization and organization of spectral data. *Methods in Ecology and Evolution*, 4(10):906-913. doi:10.1111/2041210X.12069.

See Also

Useful links:

- `https://docs.r4photobiology.info/photobiology/`
- `https://github.com/aphalo/photobiology`
- Report bugs at `https://github.com/aphalo/photobiology/issues`

Examples

```
# irradiance of the whole spectrum
irrad(sun.spct)
# photon irradiance 400 nm to 700 nm
q_irrad(sun.spct, waveband(c(400,700)))
# energy irradiance 400 nm to 700 nm
e_irrad(sun.spct, waveband(c(400,700)))
# simulating the effect of a filter on solar irradiance
e_irrad(sun.spct * yellow_gel.spct, waveband(c(400,500)))
e_irrad(sun.spct * yellow_gel.spct, waveband(c(500,700)))
# daylength
sunrise_time(lubridate::today(tzone = "Europe/Helsinki"),
             tz = "Europe/Helsinki",
             geocode = data.frame(lat = 60, lon = 25),
             unit.out = "hour")
day_length(lubridate::today(tzone = "Europe/Helsinki"),
           tz = "Europe/Helsinki",
           geocode = data.frame(lat = 60, lon = 25),
           unit.out = "hour")
# colour as seen by humans
color_of(sun.spct)
color_of(sun.spct * yellow_gel.spct)
# filter transmittance
transmittance(yellow_gel.spct)
transmittance(yellow_gel.spct, waveband(c(400,500)))
transmittance(yellow_gel.spct, waveband(c(500,700)))
```

`A.illuminant.spct` *CIE A illuminant data*

Description

A dataset containing wavelengths at a 5 nm interval (300 nm to 830 nm) and the corresponding spectral energy irradiance normalized to 1 at 560 nm. Spectrum approximates typical, domestic, tungsten-filament lighting and 'corresponds' to a black body a 2856 K. CIE standard illuminant A is intended to represent typical, domestic, tungsten-filament lighting. Original data from CIE downloaded on 2024-11-30.

Usage

```
A.illuminant.spct
```

Format

A source spectrum with 531 rows and 2 variables.

- w.length (nm)
- s.e.irrad (rel. units)

Note

This and other CIE illuminant spectra can be downloaded from `https://cie.co.at/data-tables` as .CSV files.

Author(s)

CIE

References

CIE 2018, CIE standard illuminant A - 1 nm, International Commission on Illumination (CIE), Vienna, Austria, doi:10.25039/CIE.DS.8jsxjrsn.

See Also

Other Spectral data examples: `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
A.illuminant.spct
```

A2T *Convert absorbance into transmittance*

Description

Function that converts absorbance (a.u.) into transmittance (fraction).

Usage

```
A2T(x, action, byref, ...)

## Default S3 method:
A2T(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'numeric'
A2T(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'filter_spct'
A2T(x, action = "add", byref = FALSE, ...)

## S3 method for class 'filter_mspct'
A2T(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
```

Arguments

`x`	an R object.
`action`	a character string "add" or "replace".
`byref`	logical indicating if new object will be created by reference or by copy of `x`.
`...`	not used in current version.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of `x` with a column `Tfr` added and `A` and `Afr` possibly deleted except for `w.length`. If `action = "replace"`, in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- `A2T(default)`: Default method for generic function
- `A2T(numeric)`: method for numeric vectors
- `A2T(filter_spct)`: Method for filter spectra
- `A2T(filter_mspct)`: Method for collections of filter spectra

See Also

Other quantity conversion functions: `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

---

absorbance *Absorbance*

Description

Function to calculate the mean, total, or other summary of absorbance for spectral data stored in a `filter_spct` or in an `object_spct`.

Usage

```
absorbance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## Default S3 method:
absorbance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## S3 method for class 'filter_spct'
```

```
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'object_spct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'filter_mspct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`quantity`	character string One of `"average"` or `"mean"`, `"total"`, `"contribution"`, `"contribution.pc"`, `"relative"` or `"relative.pc"`.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `absorbance(default)`: Default for generic function
- `absorbance(filter_spct)`: Specialization for filter spectra

- `absorbance(object_spct)`: Specialization for object spectra
- `absorbance(filter_mspct)`: Calculates absorbance from a `filter_mspct`
- `absorbance(object_mspct)`: Calculates absorbance from a `object_mspct`

Note

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

```
absorbance(polyester.spct, new_waveband(400,700))
absorbance(yellow_gel.spct, new_waveband(400,700))
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3))
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "average")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "total")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative.pc")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution.pc")
```

`absorptance` *Absorptance*

Description

Function to calculate the mean, total, or other summary of absorptance for spectral data stored in a `filter_spct` or in an `object_spct`. Absorptance is a different quantity than absorbance.

Usage

```
absorptance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## Default S3 method:
absorptance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## S3 method for class 'filter_spct'
absorptance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'object_spct'
absorptance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'filter_mspct'
absorptance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx"
)

## S3 method for class 'object_mspct'
absorptance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

spct	an R object.

`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`quantity`	character string One of `"average"` or `"mean"`, `"total"`, `"contribution"`, `"contribution.pc"`, `"relative"` or `"relative.pc"`.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `absorptance(default)`: Default for generic function
- `absorptance(filter_spct)`: Specialization for filter spectra
- `absorptance(object_spct)`: Specialization for object spectra
- `absorptance(filter_mspct)`: Calculates absorptance from a `filter_mspct`
- `absorptance(object_mspct)`: Calculates absorptance from a `object_mspct`

Note

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

```
absorptance(black_body.spct, new_waveband(400,500))
absorptance(white_body.spct, new_waveband(300,400))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "average")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "total")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative.pc")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution.pc")
```

`add_attr2tb` *Copy attributes from members of a* `generic_mspct`

Description

Copy metadata attributes from members of a `generic_mspct` object into a `data.frame` or a `tibble`.

Usage

```
add_attr2tb(
  tb = NULL,
  mspct,
  col.names = NULL,
  idx = "spct.idx",
  unnest = FALSE
)

when_measured2tb(mspct, tb = NULL, col.names = "when.measured", idx = NULL)

geocode2tb(mspct, tb = NULL, col.names = "geocode", idx = "spct.idx")

lonlat2tb(mspct, tb = NULL, col.names = c("lon", "lat"), idx = "spct.idx")

lon2tb(mspct, tb = NULL, col.names = "lon", idx = "spct.idx")
```

```
lat2tb(mspct, tb = NULL, col.names = "lat", idx = "spct.idx")

address2tb(mspct, tb = NULL, col.names = "address", idx = "spct.idx")

what_measured2tb(
  mspct,
  tb = NULL,
  col.names = "what.measured",
  idx = "spct.idx"
)

how_measured2tb(mspct, tb = NULL, col.names = "how.measured", idx = "spct.idx")

normalized2tb(mspct, tb = NULL, col.names = "normalized", idx = "spct.idx")

scaled2tb(mspct, tb = NULL, col.names = "scaled", idx = "spct.idx")

instr_desc2tb(mspct, tb = NULL, col.names = "instr.desc", idx = "spct.idx")

instr_settings2tb(
  mspct,
  tb = NULL,
  col.names = "instr.settings",
  idx = "spct.idx"
)

BSWF_used2tb(mspct, tb = NULL, col.names = "BSWF.used", idx = "spct.idx")

filter_properties2tb(
  mspct,
  tb = NULL,
  col.names = "filter.properties",
  idx = "spct.idx"
)

solute_properties2tb(
  mspct,
  tb = NULL,
  col.names = "solute.properties",
  idx = "spct.idx"
)

Tfr_type2tb(mspct, tb = NULL, col.names = "Tfr.type", idx = "spct.idx")

Rfr_type2tb(mspct, tb = NULL, col.names = "Rfr.type", idx = "spct.idx")

time_unit2tb(mspct, tb = NULL, col.names = "time.unit", idx = "spct.idx")
```

```
comment2tb(mspct, tb = NULL, col.names = "comment", idx = "spct.idx")

multiple_wl2tb(mspct, tb = NULL, col.names = "multiple.wl", idx = "spct.idx")
```

Arguments

tb	tibble or data.frame to which to add the data (optional).
mspct	generic_mspct or generic_spct Any collection of spectra or one or more spectra in long form.
col.names	named character vector Name(s) of metadata attributes to copy. If named, the names provide the name for the columns.
idx	character Name of the column with the names of the members of the collection of spectra.
unnest	logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.

Details

Each attribute is by default copied to a column in a tibble or a data.frame. If the argument for tb is NULL, as by default, a new tibble will be created. If an existing data.frame or tibble is passed as argument, new columns are added to it. However, the number of rows in the argument passed to tb must match the number of spectra in the argument passed to mspct. Only in the case of methods add_attr2tb() and spct_metadata() if the argument to col.names is a named vector, the names of members are used as names for the columns created. This permits setting any valid name for the new columns. If the members of the vector passed to col.names have no names, then the value is interpreted as the name of the attributes to add, and also used as name for the new column.

Valid values accepted as argument to col.names are NULL, or a vector containing one or more of the following character strings: "lon", "lat", "address", "geocode", "where.measured", "when.measured", "what.measured", "how.measured", "comment", "normalised", "normalized", "scaled", "bswf.used", "instr.desc", "instr.settings", solute.properties, "filter.properties", "Tfr.type", "Rfr.type", "time.unit".

Value

A data.frame or a tibble With the metadata attributes in separate new variables.

Note

The order of the first two arguments is reversed in add_attr2tb(), when_measured2tb(), what_measured2tb(), etc., compared to attribute query functions, such as spct_metadata, when_measured(), what_measured(), how_measured(), etc. This is to allow the use of add_attr2tb() in 'pipes' to add metadata to summaries computed at earlier steps in the pipe.

See Also

Other measurement metadata functions: getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

Examples

```
# Add attributes to irradiance
## from collection of spectra
e_irrad(sun_evening.mspct) |>
  add_attr2tb(sun_evening.mspct,
              c(when.measured = "time"))

## from spectra in long form
e_irrad(sun_evening.spct) |>
  add_attr2tb(sun_evening.spct,
              c(when.measured = "time"))

# Add attributes to transmittance
## from collection of spectra
transmittance(two_filters.mspct) |>
  add_attr2tb(two_filters.mspct, col.names = "what.measured")

transmittance(two_filters.mspct) |>
  add_attr2tb(two_filters.mspct,
              col.names = c("filter.properties", "what.measured"),
              unnest = TRUE)

# Create a new data frame
add_attr2tb(mspct = two_filters.mspct,
            idx = "filter",
            col.names = c("filter.properties", "what.measured"),
            unnest = TRUE)
```

`Afr2T` *Convert transmittance into absorptance.*

Description

Function that converts transmittance (fraction) into absorptance (fraction). If reflectance (fraction) is available, it allows conversions between internal and total absorptance.

Usage

```
Afr2T(x, action, byref, clean, ...)

## Default S3 method:
Afr2T(x, action = NULL, byref = FALSE, clean = FALSE, ...)
```

```
## S3 method for class 'numeric'
Afr2T(x, action = NULL, byref = FALSE, clean = FALSE, Rfr = NA_real_, ...)

## S3 method for class 'filter_spct'
Afr2T(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'object_spct'
Afr2T(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'filter_mspct'
Afr2T(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
Afr2T(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object
`action`	character Allowed values "replace" and "add"
`byref`	logical indicating if new object will be created by reference or by copy of x
`clean`	logical replace off-boundary values before conversion
`...`	not used in current version
`Rfr`	numeric vector. Spectral reflectance o reflectance factor. Set to zero if `x` is internal reflectance,
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of `x` with a column `Tfr` added and other columns possibly deleted except for `w.length`. If `action = "replace"`, in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- `Afr2T(default)`: Default method for generic function
- `Afr2T(numeric)`: Default method for generic function
- `Afr2T(filter_spct)`: Method for filter spectra
- `Afr2T(object_spct)`: Method for object spectra
- `Afr2T(filter_mspct)`: Method for collections of filter spectra
- `Afr2T(object_mspct)`: Method for collections of object spectra

See Also

Other quantity conversion functions: `A2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

Examples

```
T2Afr(Ler_leaf.spct)
```

`any2T` *Convert filter quantities.*

Description

Functions that convert or add related physical quantities to `filter_spct` or `object_spct` objects. transmittance (fraction) into absorptance (fraction).

Usage

```
any2T(x, action = "add", clean = FALSE)

any2A(x, action = "add", clean = FALSE)

any2Afr(x, action = "add", clean = FALSE)
```

Arguments

`x`	an filter_spct or a filter_mspct object.
`action`	character Allowed values "replace" and "add".
`clean`	logical replace off-boundary values before conversion

Details

These functions are dispatchers for A2T, Afr2T, T2A, and T2Afr. The dispatch is based on the names of the variables stored in x. They do not support in-place modification of x.

Value

A copy of x with the columns for the different quantities added or replaced. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

See Also

Other quantity conversion functions: A2T(), Afr2T(), T2A(), T2Afr(), as_quantum(), e2q(), e2qmol_multipliers(), e2quantum_multipliers(), q2e()

Examples

```
any2Afr(Ler_leaf.spct)
any2T(Ler_leaf.spct)
any2T(polyester.spct)
```

---

as.calibration_mspct *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.calibration_mspct(x, ...)

## Default S3 method:
as.calibration_mspct(x, ...)

## S3 method for class 'data.frame'
as.calibration_mspct(x, ...)

## S3 method for class 'calibration_spct'
as.calibration_mspct(x, ...)

## S3 method for class 'list'
as.calibration_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.calibration_mspct(
  x,
  w.length,
  spct.data.var = "irrad.mult",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `calibration_mspctt` object.

Methods (by class)

- `as.calibration_mspct(default)`:
- `as.calibration_mspct(data.frame)`:
- `as.calibration_mspct(calibration_spct)`:
- `as.calibration_mspct(list)`:
- `as.calibration_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

as.calibration_spct *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.calibration_spct(x, ...)

## Default S3 method:
as.calibration_spct(x, ...)
```

Arguments

x	an R object.
...	other arguments passed to "set" functions.

Value

A copy of x converted into a calibration_spct object.

Methods (by class)

- as.calibration_spct(default):

See Also

setGenericSpct

Other constructors of spectral objects: as.chroma_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.solute_spct(), as.source_spct(), source_spct()

as.chroma_mspct *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.chroma_mspct(x, ...)

## Default S3 method:
as.chroma_mspct(x, ...)

## S3 method for class 'data.frame'
as.chroma_mspct(x, ...)

## S3 method for class 'chroma_spct'
as.chroma_mspct(x, ...)

## S3 method for class 'list'
as.chroma_mspct(x, ..., ncol = 1, byrow = FALSE)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data

Value

A copy of `x` converted into a `chroma_mspct` object.

Methods (by class)

- `as.chroma_mspct(default)`:
- `as.chroma_mspct(data.frame)`:
- `as.chroma_mspct(chroma_spct)`:
- `as.chroma_mspct(list)`:

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.chroma_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.chroma_spct(x, ...)

## Default S3 method:
as.chroma_spct(x, ...)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.

Value

A copy of `x` converted into a `chroma_spct` object.

Methods (by class)

- `as.chroma_spct(default)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

`as.cps_mspct` *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.cps_mspct(x, ...)

## Default S3 method:
as.cps_mspct(x, ...)

## S3 method for class 'data.frame'
as.cps_mspct(x, ...)

## S3 method for class 'cps_spct'
as.cps_mspct(x, ...)

## S3 method for class 'list'
as.cps_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.cps_mspct(
  x,
  w.length,
  spct.data.var = "cps",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `cps_mspct` object.

Methods (by class)

- `as.cps_mspct(default)`:
- `as.cps_mspct(data.frame)`:
- `as.cps_mspct(cps_spct)`:
- `as.cps_mspct(list)`:
- `as.cps_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.cps_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.cps_spct(x, ...)

## Default S3 method:
as.cps_spct(x, ...)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.

Value

A copy of `x` converted into a `cps_spct` object.

Methods (by class)

- `as.cps_spct(default)`:

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.solute_spct(), as.source_spct(), source_spct()

---

as.filter_mspct *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.filter_mspct(x, ...)

## Default S3 method:
as.filter_mspct(x, ...)

## S3 method for class 'data.frame'
as.filter_mspct(x, Tfr.type = c("total", "internal"), strict.range = TRUE, ...)

## S3 method for class 'filter_spct'
as.filter_mspct(x, ...)

## S3 method for class 'list'
as.filter_mspct(
  x,
  Tfr.type = c("total", "internal"),
  strict.range = TRUE,
  ...,
  ncol = 1,
  byrow = FALSE
)

## S3 method for class 'matrix'
as.filter_mspct(
  x,
  w.length,
  spct.data.var = "Tfr",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`Tfr.type`	a character string, either "total" or "internal"
`strict.range`	logical Flag indicating how off-range values are handled
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > `1` how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `filter_mspct` object.

Methods (by class)

- `as.filter_mspct(default)`:
- `as.filter_mspct(data.frame)`:
- `as.filter_mspct(filter_spct)`:
- `as.filter_mspct(list)`:
- `as.filter_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

as.filter_spct *Coerce or convert into a filter spectrum*

Description

Return a possibly modified copy of an R object with its class set to a filter spectrum. In the case of conversion from a `solute_spct` object, compute the spectral quantity based on additional input from user.

Usage

```
as.filter_spct(x, ...)

## Default S3 method:
as.filter_spct(
  x,
  Tfr.type = c("total", "internal"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)

## S3 method for class 'solute_spct'
as.filter_spct(
  x,
  Tfr.type = "internal",
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  Rfr.constant = NA_real_,
  comment = NULL,
  molar.concentration = NULL,
  mass.concentration = NULL,
  path.length = 1,
  ...
)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`Tfr.type`	a character string, either `"total"` or `"internal"`.
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.
`Rfr.constant`	numeric The value of the reflection factor (/1) to be set.
`comment`	character A string to be added as a comment attribute to the object created. If not supplied, the comment will be copied from `x`.

`molar.concentration, mass.concentration`	numeric Concentration to be used to compute transmittance of the solute in solution [$mol\,m^{-3} = mmol\,dm^{-3}$ or $kg\,m^{-3} = g\,dm^{-3}$, respectively].
`path.length`	numeric The length of the light path (m) used to compute transmittance of the solute in a solution.

Value

A copy of x converted into a `filter_spct.` object.

Methods (by class)

- `as.filter_spct(default)`:
- `as.filter_spct(solute_spct)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

`as.generic_mspct` *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.generic_mspct(x, ...)

## Default S3 method:
as.generic_mspct(x, ...)

## S3 method for class 'data.frame'
as.generic_mspct(x, force.spct.class = FALSE, ...)

## S3 method for class 'generic_spct'
as.generic_mspct(x, force.spct.class = FALSE, ...)

## S3 method for class 'list'
as.generic_mspct(x, force.spct.class = FALSE, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.generic_mspct(
  x,
  w.length,
  member.class,
  spct.data.var,
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)

mat2mspct(
  x,
  w.length,
  member.class,
  spct.data.var,
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`force.spct.class`	logical indicating whether to change the class of members to `generic_spct` or retain the existing class.
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > `1` how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`member.class`	character The name of the class of the individual spectra to be constructed.
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `generic_mspct` object.

Methods (by class)

- `as.generic_mspct(default)`:
- `as.generic_mspct(data.frame)`:
- `as.generic_mspct(generic_spct)`:
- `as.generic_mspct(list)`:
- `as.generic_mspct(matrix)`:

Note

Members of `generic_mspct` objects can be heterogeneous: they can belong to any class derived from `generic_spct` and class is not enforced. When `x` is a list of data frames `force.spct.class = TRUE` needs to be supplied. When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.generic_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.generic_spct(x, ...)

## Default S3 method:
as.generic_spct(x, ...)
```

Arguments

`x`	an R object
`...`	other arguments passed to "set" functions

Value

A copy of `x` converted into a `generic_spct` object.

Methods (by class)

- `as.generic_spct(default)`:

See Also

setGenericSpct

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

`as.matrix-mspct` *Coerce a collection of spectra into a matrix*

Description

Convert an object of class `generic_mspct` or a derived class into an R matrix with wavelengths saved as an attribute and spectral data in rows or columns.

Usage

```
## S3 method for class 'generic_mspct'
as.matrix(x, spct.data.var, byrow = attr(x, "mspct.byrow"), ...)

mspct2mat(x, spct.data.var, byrow = attr(x, "mspct.byrow"), ...)
```

Arguments

`x`	generic_mspct object.
`spct.data.var`	character The name of the variable containing the spectral data.
`byrow`	logical. If FALSE (the default) the matrix is filled with the spectra stored by columns, otherwise the matrix is filled by rows.
`...`	currently ignored.

Warning!

This conversion preserves the spectral data but discards almost all the metadata contained in the spectral objects. In other words a matrix created with this function cannot be used to recreate the original object unless the same metadata is explicitly supplied when converting the matrix into new collection of spectra.

Note

Only collections of spectra containing spectra with exactly the same `w.length` values can by converted. If needed, the spectra can be re-expressed before attempting the conversion to a matrix.

as.object_mspct *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.object_mspct(x, ...)

## Default S3 method:
as.object_mspct(x, ...)

## S3 method for class 'data.frame'
as.object_mspct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...
)

## S3 method for class 'object_spct'
as.object_mspct(x, ...)

## S3 method for class 'list'
as.object_mspct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...,
  ncol = 1,
  byrow = FALSE
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`Tfr.type`	a character string, either "total" or "internal"
`Rfr.type`	a character string, either "total" or "specular"
`strict.range`	logical Flag indicating how off-range values are handled
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data

Value

A copy of `x` converted into a `object_mspct` object.

Methods (by class)

- `as.object_mspct(default)`:
- `as.object_mspct(data.frame)`:
- `as.object_mspct(object_spct)`:
- `as.object_mspct(list)`:

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.object_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.object_spct(x, ...)

## Default S3 method:
as.object_spct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`Tfr.type`	a character string, either `"total"` or `"internal"`.
`Rfr.type`	a character string, either `"total"` or `"specular"`.
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.

Value

A copy of x converted into a object_spct object.

Methods (by class)

- `as.object_spct(default)`:

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.solute_spct(), as.source_spct(), source_spct()

---

as.raw_mspct   *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.raw_mspct(x, ...)

## Default S3 method:
as.raw_mspct(x, ...)

## S3 method for class 'data.frame'
as.raw_mspct(x, ...)

## S3 method for class 'raw_spct'
as.raw_mspct(x, ...)

## S3 method for class 'list'
as.raw_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.raw_mspct(
  x,
  w.length,
  spct.data.var = "counts",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `raw_mspct` object.

Methods (by class)

- `as.raw_mspct(default)`:
- `as.raw_mspct(data.frame)`:
- `as.raw_mspct(raw_spct)`:
- `as.raw_mspct(list)`:
- `as.raw_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.raw_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.raw_spct(x, ...)

## Default S3 method:
as.raw_spct(x, ...)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.

Value

A copy of `x` converted into a `raw_spct` object.

Methods (by class)

- `as.raw_spct(default)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

`as.reflector_mspct` *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.reflector_mspct(x, ...)

## Default S3 method:
as.reflector_mspct(x, ...)

## S3 method for class 'data.frame'
as.reflector_mspct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...
)

## S3 method for class 'reflector_spct'
as.reflector_mspct(x, ...)

## S3 method for class 'list'
as.reflector_mspct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...,
  ncol = 1,
  byrow = FALSE
)

## S3 method for class 'matrix'
as.reflector_mspct(
  x,
  w.length,
  spct.data.var = "Rfr",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`Rfr.type`	a character string, either "total" or "specular"
`strict.range`	logical Flag indicating how off-range values are handled
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data

`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `reflector_mspct` object.

Methods (by class)

- `as.reflector_mspct(default)`:
- `as.reflector_mspct(data.frame)`:
- `as.reflector_mspct(reflector_spct)`:
- `as.reflector_mspct(list)`:
- `as.reflector_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.reflector_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.reflector_spct(x, ...)

## Default S3 method:
as.reflector_spct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`Rfr.type`	a character string, either `"total"` or `"specular"`.
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.

Value

A copy of `x` converted into a `reflector_spct` object.

Methods (by class)

- `as.reflector_spct(default)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

`as.response_mspct` *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.response_mspct(x, ...)

## Default S3 method:
as.response_mspct(x, ...)

## S3 method for class 'data.frame'
as.response_mspct(x, time.unit = "second", ...)

## S3 method for class 'response_spct'
as.response_mspct(x, ...)

## S3 method for class 'list'
as.response_mspct(x, time.unit = "second", ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.response_mspct(
  x,
  w.length,
  spct.data.var = "s.e.response",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`time.unit`	character A string, "second", "day" or "exposure"
`ncol`	integer Number of ’virtual’ columns in data
`byrow`	logical If `ncol` > `1` how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `response_mspct` object.

Methods (by class)

- `as.response_mspct(default)`:
- `as.response_mspct(data.frame)`:
- `as.response_mspct(response_spct)`:
- `as.response_mspct(list)`:
- `as.response_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

`as.response_spct` *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.response_spct(x, ...)

## Default S3 method:
as.response_spct(x, time.unit = "second", ...)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`time.unit`	character string indicating the time unit used for spectral irradiance or exposure (`"second"`, `"day"` or `"exposure"`) or an object of class duration as defined in package lubridate.

Value

A copy of `x` converted into a `response_spct` object.

Methods (by class)

- `as.response_spct(default)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.solute_spct()`, `as.source_spct()`, `source_spct()`

---

as.solute_mspct *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.solute_mspct(x, ...)

## Default S3 method:
as.solute_mspct(x, ...)

## S3 method for class 'data.frame'
as.solute_mspct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  strict.range = TRUE,
  ...
)

## S3 method for class 'solute_spct'
as.solute_mspct(x, ...)

## S3 method for class 'list'
as.solute_mspct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  strict.range = TRUE,
  ...,
  ncol = 1,
  byrow = FALSE
)

## S3 method for class 'matrix'
```

```
as.solute_mspct(
  x,
  w.length,
  spct.data.var = "K.mole",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`K.type`	a character string, either "attenuation", "absorption" or "scattering"
`strict.range`	logical Flag indicating how off-range values are handled
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`w.length`	numeric A vector of wavelength values sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `filter_mspct` object.

Methods (by class)

- `as.solute_mspct(default)`:
- `as.solute_mspct(data.frame)`:
- `as.solute_mspct(solute_spct)`:
- `as.solute_mspct(list)`:
- `as.solute_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()

as.solute_spct *Coerce to a solute spectrum*

Description

Return a possibly modified copy of an R object with its class set to solute_spct (a solute spectrum). In the case of conversion from a filter_spct object, compute spectral molar attenuation based on additional input from user.

Usage

```
as.solute_spct(x, ...)

## Default S3 method:
as.solute_spct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)

## S3 method for class 'filter_spct'
as.solute_spct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  name = NA_character_,
  mass = NA_character_,
  formula = NULL,
  structure = grDevices::as.raster(matrix()),
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  comment = NULL,
  molar.concentration = NULL,
  mass.concentration = NULL,
  path.length = 1,
  ...
)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`K.type`	a character string, one of `"attenuation"`, `"absorption"` or `"scattering"`.
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.
`name`, `solvent.name`	character The names of the substance and of the solvent. A named character vector, with member names such as "IUPAC" for the authority.
`mass`	numeric The mass in Dalton (Da = g/mol).
`formula`	character The molecular formula.
`structure`	raster A bitmap of the structure.
`ID`, `solvent.ID`	character The IDs of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChen" for the authority.
`comment`	character A string to be added as a comment attribute to the object created. If not supplied, the comment will be copied from `x`.
`molar.concentration`, `mass.concentration`	numeric Concentration to be used to compute transmittance of the solute in solution [$mol\, m^{-3} = mmol\, dm^{-3}$ or $kg\, m^{-3} = g\, dm^{-3}$, respectively].
`path.length`	numeric The length of the light path (m) used to compute transmittance of the solute in a solution.

Value

A copy of `x` converted into a `solute_spct` object.

Methods (by class)

- `as.solute_spct(default)`:
- `as.solute_spct(filter_spct)`:

See Also

`setSoluteSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.source_spct()`, `source_spct()`

`as.source_mspct` *Coerce to a collection-of-spectra*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.source_mspct(x, ...)

## Default S3 method:
as.source_mspct(x, ...)

## S3 method for class 'data.frame'
as.source_mspct(
  x,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)

## S3 method for class 'source_spct'
as.source_mspct(x, ...)

## S3 method for class 'list'
as.source_mspct(
  x,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...,
  ncol = 1,
  byrow = FALSE
)

## S3 method for class 'matrix'
as.source_mspct(
  x,
  w.length,
  spct.data.var = "s.e.irrad",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

Arguments

`x`	a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
`...`	passed to individual spectrum object constructor
`time.unit`	character A string, "second", "day" or "exposure"
`bswf.used`	character
`strict.range`	logical Flag indicating how off-range values are handled
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`w.length`	numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).
`spct.data.var`	character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
`multiplier`	numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
`spct.names`	character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `source_mspct` object.

Methods (by class)

- `as.source_mspct(default)`:
- `as.source_mspct(data.frame)`:
- `as.source_mspct(source_spct)`:
- `as.source_mspct(list)`:
- `as.source_mspct(matrix)`:

Note

When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `split2mspct()`, `subset2mspct()`

as.source_spct — *Coerce to a spectrum*

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

```
as.source_spct(x, ...)

## Default S3 method:
as.source_spct(
  x,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)
```

Arguments

`x`	an R object.
`...`	other arguments passed to "set" functions.
`time.unit`	character string indicating the time unit used for spectral irradiance or exposure (`"second"`, `"day"` or `"exposure"`) or an object of class duration as defined in package lubridate.
`bswf.used`	character A string indicating the BSWF used, if any, for spectral effective irradiance or exposure (`"none"` or the name of the BSWF).
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.

Value

A copy of `x` converted into a `source_spct` object.

Methods (by class)

- `as.source_spct(default)`:

See Also

`setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `source_spct()`

as_energy — *Convert spectral photon irradiance into spectral energy irradiance*

Description

Convert a spectral photon irradiance [$mol\ s^{-1}\ m^{-2}\ nm^{-1}$] into a spectral energy irradiance [$W\ m^{-2}\ nm^{-1}$].

Usage

```
as_energy(w.length, s.qmol.irrad)
```

Arguments

`w.length`	numeric vector of wavelengths [nm]).
`s.qmol.irrad`	numeric vector of spectral photon irradiance values.

Value

A numeric vector of spectral (energy) irradiances.

See Also

Other low-level functions operating on numeric vectors.: `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.spct, as_energy(w.length, s.q.irrad))
```

as_quantum — *Convert spectral energy irradiance into spectral photon irradiance*

Description

Convert spectral energy irradiance [W m-2 nm-1] into spectral photon irradiance expressed as number of photons [s-1 m-2 nm-1]

Usage

```
as_quantum(w.length, s.e.irrad)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`s.e.irrad`	numeric vector of spectral (energy) irradiance values.

Value

A numeric vector of spectral photon irradiances.

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

Examples

```
with(sun.data, as_quantum(w.length, s.e.irrad))
```

`as_quantum_mol` *Convert spectral energy irradiance into spectral photon irradiance*

Description

Convert spectral energy irradiance $[W\,m^{-2}\,nm^{-1}]$ into a spectral photon irradiance expressed in number of molds of photons $[mol\,s^{-1}\,m^{-2}\,nm^{-1}]$.

Usage

```
as_quantum_mol(w.length, s.e.irrad)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`s.e.irrad`	numeric vector of spectral (energy) irradiance values.

Value

a numeric vector of spectral photon irradiances.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data, as_quantum_mol(w.length, s.e.irrad))
```

`average_spct` *Average spectral data.*

Description

This function gives the result of integrating spectral data over wavelengths and dividing the result by the spread or span of the wavelengths.

Usage

```
average_spct(spct)
```

Arguments

`spct`	generic_spct

Value

One or more numeric values with no change in scale factor: e.g. [W m-2 nm-1] -> [W m-2 nm-1]. Each value in the returned vector corresponds to a variable in the spectral object, except for wavelength.

Examples

```
average_spct(sun.spct)
```

`beesxyzCMF.spct` *Honeybee xyz chromaticity colour matching function data*

Description

A dataset containing wavelengths at a 5 nm interval (300 nm to 700 nm) and the corresponding x, y, and z chromaticity coordinates. Original data from XXX.

A chroma_spct object with variables as follows:

Usage

```
beesxyzCMF.spct
```

Format

A data frame with 81 rows and 4 variables

Details

- w.length (nm)
- x
- y
- z

See Also

Other Visual response data examples: `ciev10.spct`, `ciev2.spct`, `ciexyzCC10.spct`, `ciexyzCC2.spct`, `ciexyzCMF10.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

`black_body.spct` *Theoretical optical bodies*

Description

Datasets for a hypothetical objects with transmittance 0/1 (0%), reflectance 0/1 (0%), with transmittance 0/1 (0%), reflectance 1/1 (100%), and with with transmittance 1/1 (100%), reflectance 0/1 (0%).

Format

A `object_spct` object with 4 rows and 3 variables

Details

- w.length (nm)
- Tfr (0..1)
- Rfr (0..1)

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

c *Combine collections of spectra*

Description

Combine two or more generic_mspct objects into a single object.

Usage

```
## S3 method for class 'generic_mspct'
c(..., recursive = FALSE, ncol = 1, byrow = FALSE)
```

Arguments

`...`	one or more generic_mspct objects to combine.
`recursive`	logical ignored as nesting of collections of spectra is not supported.
`ncol`	numeric Virtual number of columns
`byrow`	logical When object has two dimensions, how to map member objects to columns and rows.

Value

A collection of spectra object belonging to the most derived class shared among the combined objects.

calc_multipliers *Spectral weights*

Description

Calculate multipliers for selecting a range of wavelengths and optionally applying a biological spectral weighting function (BSWF) and wavelength normalization. This function returns numeric multipliers that can be used to select a waveband and apply a weight.

Usage

```
calc_multipliers(
  w.length,
  w.band,
  unit.out = "energy",
  unit.in = "energy",
  use.cached.mult = FALSE,
  fill = 0
)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`w.band`	waveband object.
`unit.out`	character One of "photon" or "energy", default is "energy".
`unit.in`	character One of "photon" or "energy", default is "energy".
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`fill`	numeric If `fill` = `NA` then values returned for wavelengths outside the range of the waveband are set to `NA`.

Value

a numeric vector of multipliers of the same length as `w.length`.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data,
     calc_multipliers(w.length = w.length,
                      w.band = new_waveband(400,700),
                      unit.out = "photon"))
with(sun.data,
     calc_multipliers(w.length = w.length,
                      w.band = new_waveband(400,700),
                      unit.out = "photon"),
                      use.cached.mult = TRUE)
```

`calc_source_output` *Scaled and/or interpolated light-source spectral output*

Description

Values calculated by interpolation from user-supplied spectral emission data or by name for light source data included in the packages photobiologySun, photobiologyLamps, or photobiologyLEDs, optionally re-scaling the spectral data values.

Usage

```
calc_source_output(
  w.length.out,
  w.length.in,
  s.irrad.in,
  unit.in = "energy",
  scaled = NULL,
  fill = NA,
  ...
)
```

Arguments

`w.length.out`	numeric vector of wavelengths (nm) for output.
`w.length.in`	numeric vector of wavelengths (nm) for input.
`s.irrad.in`	numeric vector of spectral transmittance value (fractions or percent).
`unit.in`	a character string "energy" or "photon".
`scaled`	NULL, "peak", "area"; div ignored if !is.null(scaled).
`fill`	if NA, no extrapolation is done, and NA is returned for wavelengths outside the range of the input. If NULL then the tails are deleted. If 0 then the tails are set to zero.
`...`	Additional arguments passed to `spline` if called.

Value

a source_spct with three numeric vectors with wavelength values (w.length), scaled and interpolated spectral energy irradiance (s.e.irrad), scaled and interpolated spectral photon irradiance values (s.q.irrad).

Note

This is a convenience function that adds no new functionality but makes it a little easier to plot lamp spectral emission data consistently. It automates interpolation, extrapolation/trimming and scaling.

Examples

```
with(sun.data,
     calc_source_output(290:1100,
                        w.length.in = w.length,
                        s.irrad.in = s.e.irrad)
    )
```

ccd.spct *Spectral response of a back-thinned CCD image sensor.*

Description

A dataset containing wavelengths at a 1 nm interval and spectral response as quantum efficiency for CCD sensor type S11071/S10420 from Hamamatsu (measured without a quartz window). These vectors are frequently used as sensors in high-UV-sensitivity vector spectrometers. Data digitized from manufacturer's data sheet. The original data is expressed as percent quantum efficiency with a value of 77% at the peak. The data have been re-expressed as fractions of one.

Usage

```
ccd.spct
```

Format

A `response_spct` object with 186 rows and 2 variables

Details

- w.length (nm).
- s.q.response (fractional quantum efficiency)

References

Hamamatsu (2014) Datasheet: CCD Image Sensors S11071/S10420-01 Series. Hamamatsu Photonics KK, Hamamatsu, City. http://www.hamamatsu.com/jp/en/S11071-1004.html. Visited 2017-12-15.

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
ccd.spct
```

checkTimeUnit — *Check the "time.unit" attribute of an existing source_spct object*

Description

Function to read the "time.unit" attribute

Usage

```
checkTimeUnit(x)
```

Arguments

`x`	a source_spct object

Value

x possibly with the `time.unit` attribute modified

Note

if x is not a `source_spct` or a `response_spct` object, NA is returned

See Also

Other time attribute functions: `convertThickness()`, `convertTimeUnit()`, `getTimeUnit()`, `setTimeUnit()`

check_spct — *Check validity of spectral objects*

Description

Check that an R object contains the expected data members and within range values in them. For wavelengths also check if ordered and if unique or not.

Usage

```
check_spct(x, byref, strict.range, force = FALSE, ...)

## Default S3 method:
check_spct(x, byref = FALSE, strict.range = NA, force = FALSE, ...)

## S3 method for class 'generic_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = NA,
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'calibration_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'raw_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'cps_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'filter_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'solute_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'reflector_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'object_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'response_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = NA,
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'source_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)
```

```
## S3 method for class 'chroma_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)
```

Arguments

`x`	An R object
`byref`	logical indicating if new object will be created by reference or by copy of `x`
`strict.range`	logical indicating whether off-range values result in an error instead of a warning, with `NA` a message is issued on failure, and with `NULL` the range test is skipped.
`force`	logical If `TRUE` check is done even if checks are disabled.
`...`	additional parameters possible in derived methods
`multiple.wl`	numeric Maximum number of repeated w.length entries with same value. If `NULL` skip check of ordering and multiple wavelengths.

Details

These methods are exported and can be called by user code if needed, for example, when the checks have been disabled by setting an R option with `disable_check_spct`.

Methods (by class)

- `check_spct(default)`: Default for generic function.
- `check_spct(generic_spct)`: Specialization for generic_spct.
- `check_spct(calibration_spct)`: Specialization for calibration_spct.
- `check_spct(raw_spct)`: Specialization for raw_spct.
- `check_spct(cps_spct)`: Specialization for cps_spct.
- `check_spct(filter_spct)`: Specialization for filter_spct.
- `check_spct(solute_spct)`: Specialization for solute_spct.
- `check_spct(reflector_spct)`: Specialization for reflector_spct.
- `check_spct(object_spct)`: Specialization for object_spct.
- `check_spct(response_spct)`: Specialization for response_spct.
- `check_spct(source_spct)`: Specialization for source_spct.
- `check_spct(chroma_spct)`: Specialization for chroma_spct.

See Also

Other data validity check functions: `check_spectrum()`, `check_w.length()`, `enable_check_spct()`

Examples

```
check_spct(sun.spct)

check_spct(sun.spct)
# try(check_spct(-sun.spct))
# try(check_spct((sun.spct[1, "w.length"] <- 1000)))
```

`check_spectrum` *Sanity check a spectrum*

Description

Checks spectral irradiance data in `numeric` vectors for compliance with assumptions used in calculations.

Usage

```
check_spectrum(w.length, s.irrad)
```

Arguments

`w.length`	numeric vector of wavelengths $[nm]$.
`s.irrad`	numeric Corresponding vector of spectral (energy) irradiances $[W\,m^{-2}\,nm^{-1}]$.

Value

A single `logical` value indicating whether test was passed or not

See Also

Other data validity check functions: `check_spct()`, `check_w.length()`, `enable_check_spct()`

Examples

```
with(sun.data, check_spectrum(w.length, s.e.irrad))
```

`check_w.length` *Sanity check of wavelengths (internal function).*

Description

This function checks a w.length vector for compliance with assumptions expected for valid calculations.

Usage

```
check_w.length(w.length)
```

Arguments

`w.length`	numeric array of wavelength (nm)

Value

a single logical value indicating whether test was passed or not

See Also

Other data validity check functions: `check_spct()`, `check_spectrum()`, `enable_check_spct()`

Examples

```
with(sun.data, photobiology:::check_w.length(w.length))
```

`check_wl_stepsize` *Check consistency of wavelength step size*

Description

Check the spread of wavelength step sizes in an ordered `numeric` vector, or in the `"w.length"` column of a spectral object containing a single spectrum.

Usage

```
check_wl_stepsize(x, span = Inf, na.rm = FALSE)
```

Arguments

`x`	numeric vector.
`span`	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width span centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. `span = NULL` extends the span to the whole length of `x`.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for peaks.

Details

As the search for peaks uses a window based on a fixed number of observations at neighbouring wavelengths, if the wavelength step between observations varies drastically, the window expressed in nanometres of wavelength becomes very irregular. With the default `span = 5` in `peaks()`, `valleys()`, and `wls_at_target()` the search in most cases still works for "thinned" spectra, and the check is skipped. With `spikes()` and `despike()` methods the check is always done as these methods do not override `span = Inf`.

The typical case when the step can vary strongly are spectra returned by `thin_wl()`. As when using default arguments, including `span = 21`, `thin_wl()` retains the original local maxima and global maximum, and a reasonably narrow wavelength maximum step a call to peaks with `span = NULL` or `span = 5` *tends* to discover the original peaks missing at most a few.

Value

A logical `TRUE` is returned invisibly if check is passed and otherwise `FALSE` with a warning. A warning is issued on failure as a side effect.

Examples

```
check_wl_stepsize(sun.spct)
check_wl_stepsize(1:20, 30)
```

`ciev10.spct` *Linear energy CIE 2008 luminous efficiency function 10 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding response values for a 10 degrees target. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- s.e.response

Usage

```
ciev10.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also

Other Visual response data examples: `beesxyzCMF.spct`, `ciev2.spct`, `ciexyzCC10.spct`, `ciexyzCC2.spct`, `ciexyzCMF10.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

Examples

```
ciev10.spct
```

`ciev2.spct` *Linear energy CIE 2008 luminous efficiency function 2 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding response values for a 2 degrees target. Original data from `http://www.cvrl.org/` downloaded on 2014-04-29 The variables are as follows:

Usage

```
ciev2.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Details

- w.length (nm)
- s.e.response

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from `https://cie.co.at/data-tables`.

Author(s)

CIE

See Also

Other Visual response data examples: `beesxyzCMF.spct`, `ciev10.spct`, `ciexyzCC10.spct`, `ciexyzCC2.spct`, `ciexyzCMF10.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

Examples

```
ciev2.spct
```

`ciexyzCC10.spct` *CIE xyz chromaticity coordinates (CC) 10 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z chromaticity coordinates. Derived from proposed CIE 2006 standard. Original data from `http://www.cvrl.org/` downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

```
ciexyzCC10.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from `https://cie.co.at/data-tables`.

Author(s)

CIE

See Also

Other Visual response data examples: `beesxyzCMF.spct`, `ciev10.spct`, `ciev2.spct`, `ciexyzCC2.spct`, `ciexyzCMF10.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

Examples

```
ciexyzCC10.spct
```

`ciexyzCC2.spct` *CIE xyz chromaticity coordinates 2 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z chromaticity coordinates. According to proposed CIE 2006 standard. Original data from `http://www.cvrl.org/` downloaded on 2014-04-28 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

```
ciexyzCC2.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from `https://cie.co.at/data-tables`.

Author(s)

CIE

See Also

Other Visual response data examples: `beesxyzCMF.spct`, `ciev10.spct`, `ciev2.spct`, `ciexyzCC10.spct`, `ciexyzCMF10.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

Examples

```
ciexyzCC2.spct
```

`ciexyzCMF10.spct` *Linear energy CIE xyz colour matching function (CMF) 10 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z 10 degrees CMF values. Derived from proposed CIE 2006 standard. Original data from `http://www.cvrl.org/` downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

```
ciexyzCMF10.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from `https://cie.co.at/data-tables`.

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from `https://cie.co.at/data-tables`.

Author(s)

CIE

See Also

Other Visual response data examples: `beesxyzCMF.spct`, `ciev10.spct`, `ciev2.spct`, `ciexyzCC10.spct`, `ciexyzCC2.spct`, `ciexyzCMF2.spct`, `cone_fundamentals10.spct`

Examples

```
ciexyzCMF10.spct
```

ciexyzCMF2.spct *Linear energy CIE xyz colour matching function (CMF) 2 deg data*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z 2 degrees CMF values. Derived from proposed CIE 2006 standard. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

```
ciexyzCMF2.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from https://cie.co.at/data-tables.

Author(s)

CIE

See Also

Other Visual response data examples: beesxyzCMF.spct, ciev10.spct, ciev2.spct, ciexyzCC10.spct, ciexyzCC2.spct, ciexyzCMF10.spct, cone_fundamentals10.spct

Examples

```
ciexyzCMF2.spct
```

`class_spct` *Query which is the class of a spectrum*

Description

Extract class information from a generic spectrum.

Usage

```
class_spct(x)
```

Arguments

`x`	any R object

Details

The value returned is equivalent to the set intersection of the value returned by `class(x)` and the value returned by `spct_classes`, but preserving the order of the members of the character vector.

Value

A character vector containing all matching xxxx.spct S3 classes.

Examples

```
class_spct(sun.spct)
class(sun.spct)
```

`clean` *Clean (=replace) off-range values in a spectrum*

Description

These functions implement the equivalent of replace() but for spectral objects instead of vectors.

Usage

```
clean(x, range, range.s.data, fill, ...)

## Default S3 method:
clean(x, range, range.s.data, fill, ...)

## S3 method for class 'source_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
clean(
  x,
  range = x,
  range.s.data = NULL,
  fill = range.s.data,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
clean(x, range = x, range.s.data = c(0, 1), fill = range.s.data, ...)

## S3 method for class 'solute_spct'
clean(x, range = x, range.s.data = c(0, NA), fill = range.s.data, ...)

## S3 method for class 'object_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, 1),
  fill = range.s.data,
  min.Afr = NULL,
  ...
)

## S3 method for class 'response_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'cps_spct'
clean(x, range = x, range.s.data = c(0, NA), fill = range.s.data, ...)

## S3 method for class 'raw_spct'
```

```
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  fill = range.s.data,
  ...
)

## S3 method for class 'generic_spct'
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  fill = range.s.data,
  col.names,
  ...
)

## S3 method for class 'source_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
clean(
  x,
  range = NULL,
  range.s.data = NULL,
  fill = range.s.data,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, 1),
  fill = range.s.data,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, 1),
  fill = range.s.data,
  min.Afr = NULL,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  fill = range.s.data,
  col.names,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object
`range`	numeric vector of wavelengths
`range.s.data`	numeric vector of length two giving the allowable range for the spectral data.
`fill`	numeric vector of length 1 or 2, giving the replacement values to use at each extreme of the range.
`...`	currently ignored
`unit.out`	character string with allowed values "energy", and "photon", or its alias "quantum"
`qty.out`	character string with allowed values "energy", and "photon", or its alias "quantum"
`min.Afr`	numeric Gives the minimum value accepted for the computed absorptance. The default `NULL` sets a valid value (Afr >= 0) with a warning. If an integer value is passed to `digits` values are adjusted silently.
`col.names`	character The name of the variable to clean
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of `x`, possibly with some of the spectral data values replaced by the value passed to `fill`.

Methods (by class)

- `clean(default)`: Default for generic function
- `clean(source_spct)`: Replace off-range values in a source spectrum
- `clean(filter_spct)`: Replace off-range values in a filter spectrum
- `clean(reflector_spct)`: Replace off-range values in a reflector spectrum
- `clean(solute_spct)`: Replace off-range values in a solute spectrum
- `clean(object_spct)`: Replace off-range values in an object spectrum
- `clean(response_spct)`: Replace off-range values in a response spectrum
- `clean(cps_spct)`: Replace off-range values in a counts per second spectrum
- `clean(raw_spct)`: Replace off-range values in a raw counts spectrum
- `clean(generic_spct)`: Replace off-range values in a generic spectrum
- `clean(source_mspct)`:
- `clean(filter_mspct)`:
- `clean(reflector_mspct)`:
- `clean(object_mspct)`:
- `clean(solute_mspct)`:
- `clean(response_mspct)`:
- `clean(cps_mspct)`:
- `clean(raw_mspct)`:
- `clean(generic_mspct)`:

Note

In the case of `object_spct` objects, cleaning is done first on the Rfr and Tfr columns and subsequently Afr estimated and if needed half of deviation of Afr from the expected minimum value subtracted from each of Rfr and Tfr.

`clear.spct` *Theoretical spectrum of clear and apaque materials*

Description

Dataset for hypothetical objects with transmittance 1/1 (100%) and transmittance 0/1 (0%)

Usage

```
clear.spct

opaque.spct
```

Format

A `filter_spct` object with 4 rows and 2 variables

An object of class `filter_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 4 rows and 2 columns.

Details

- w.length (nm).
- Tfr (0..1)

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
clear.spct
opaque.spct
```

`clip_wl` *Clip head and/or tail of a spectrum*

Description

Clip head and tail of a spectrum based on wavelength limits, no interpolation used at range boundaries.

Usage

```
clip_wl(x, range, ...)

## Default S3 method:
clip_wl(x, range, ...)

## S3 method for class 'generic_spct'
clip_wl(x, range = NULL, ...)

## S3 method for class 'generic_mspct'
clip_wl(x, range = NULL, expand = TRUE, ...)

## S3 method for class 'waveband'
clip_wl(x, range = NULL, ...)
```

```
## S3 method for class 'list'
clip_wl(x, range = NULL, ...)
```

Arguments

`x`	an R object.
`range`	a numeric vector of length two, or any other object for which function `range()` will return range of wavelengths expressed in nanometres.
`...`	ignored (possibly used by derived methods).
`expand`	logical Expand or not members containing spectra in long form.

Value

a spectrum object or a collection of spectral objects of the same class as `x` with wavelength heads and tails clipped.

Methods (by class)

- `clip_wl(default)`: Default for generic function
- `clip_wl(generic_spct)`: Clip an object of class "generic_spct" or derived.
- `clip_wl(generic_mspct)`: Clip an object of class "generic_mspct" or derived.
- `clip_wl(waveband)`: Clip an object of class "waveband".
- `clip_wl(list)`: Clip a list (of objects of class "waveband").

Note

The condition tested is `wl` >= `range[1]` & `wl` < `(range[2] + 1e-13)`.

See Also

Other trim functions: `trim_spct()`, `trim_waveband()`, `trim_wl()`

Examples

```
clip_wl(sun.spct, range = c(400, 500))
clip_wl(sun.spct, range = c(NA, 500))
clip_wl(sun.spct, range = c(400, NA))
```

collect2mspct *Form a new collection*

Description

Form a collection of spectra from separate objects in the parent frame of the call.

Usage

```
collect2mspct(
  .list = NULL,
  pattern = "*\\.spct$",
  collection.class = NULL,
  ...
)
```

Arguments

`.list`	list of R objects
`pattern`	character an optional regular expression, ignored if `.list` is not `NULL`.
`collection.class`	character vector
`...`	additional named arguments passed down to the collection constructor.

Details

This is a convenience function that simplifies the creation of collections from existing objects of class `generic_spct` or a derived class. A list of objects con be passed as argument, or a search pattern. If a list is passed, no search is done. If `collection.class` is `NULL`, then all objects of class `generic_spct` or of a class derived from it are added to the collection. If objects of only one derived class are to be collected this class or that of the matching collection should be passed as argument to `collection.class`. Objects of other R classes are silently discarded, which simplifies the specification of search patterns. By default, i.e., if `collection.class` is `NULL`, if all the objects collected belong to the same class then the corresponding collection class will be returned, otherwise a `generic_mspct` object with heterogeneous members will be returned. To force the return of a `generic_mspct` even when the collected spectra all belong to the same class, pass `generic_mspct` as argument to `collection.class`. If the argument to `collection.class` is a vector containing two of more class names, only the matching spectra will be collected, and a `generic_mspct` will be returned. The returned object is created with the constructor for the class, and validated.

Value

By default a collection of spectra.

See Also

Other experimental utility functions: drop_user_cols(), thin_wl(), uncollect2spct()

Examples

```
collect2mspct() # returns empty generic_mspct object

sun1.spct <- sun.spct
sun2.spct <- sun.spct
kk.spct <- 10:30 # ignored
collect2mspct()
collect2mspct(collection.class = "generic_mspct")

pet1.spct <- polyester.spct
collect2mspct()
collect2mspct(collection.class = "source_mspct")
collect2mspct(collection.class = "filter_mspct")
collect2mspct(collection.class = "response_mspct")
```

---

color_of *Color of an object*

Description

Equivalent RGB color of an object such as a spectrum, wavelength or waveband.

Usage

```
color_of(x, ...)

## Default S3 method:
color_of(x, ...)

## S3 method for class 'numeric'
color_of(x, type = "CMF", chroma.type = type, ...)

## S3 method for class 'list'
color_of(x, short.names = TRUE, type = "CMF", chroma.type = type, ...)

## S3 method for class 'waveband'
color_of(x, short.names = TRUE, type = "CMF", chroma.type = type, ...)

## S3 method for class 'source_spct'
color_of(x, type = "CMF", chroma.type = type, ...)

## S3 method for class 'source_mspct'
color_of(x, ..., idx = "spct.idx")
```

```
colour_of(x, ...)

color(x, ...)

fast_color_of_wl(x, type = "CMF", ...)

fast_color_of_wb(x, type = "CMF", ...)
```

Arguments

`x`	an R object.
`...`	ignored (possibly used by derived methods).
`type`, `chroma.type`	character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class `chroma_spct` for any other trichromic visual system.
`short.names`	logical indicating whether to use short or long names for wavebands
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

A color definition in hexadecimal format as a `character` string of 7 characters, "#" followed by the red, blue, and green values in hexadecimal (scaled to 0 ... 255). In the case of the specialization for `list`, a list of such definitions is returned. In the case of a collection of spectra, a `data.frame` with one column with such definitions and by default an additional column with names of the spectra as index. In case of missing input the returned value is `NA`.

Methods (by class)

- `color_of(default)`: Default method (returns always "black").
- `color_of(numeric)`: Method that returns Color definitions corresponding to numeric values representing a wavelengths in nm.
- `color_of(list)`: Method that returns Color of elements in a list.
- `color_of(waveband)`: Color at midpoint of a `waveband` object.
- `color_of(source_spct)`:
- `color_of(source_mspct)`:

Deprecated

Use of color() is deprecated as this wrapper function may be removed in future versions of the package because of name clashes. Use color_of() instead.

Note

The specialization of `color_of()` for `numeric` and function `fast_color_of_wl()` accept both positive and negative values in `x` as long as all values have the same sign. This makes its use in 'ggspectra' simpler as the reverse scale transform changes the sign of the data. This should be considered a temporary fix.

When `x` is a list but not a waveband, if a method `color_of` is not available for the class of each element of the list, then `color_of.default` will be called.

Function `fast_color_of_wl()` should be used only when high performance is needed. It speeds up performance by rounding the wavelength values in the numeric vector passed as argument to `x` and then retrieves the corresponding pre-computed color definitions if `type` is either `"CMF"` or `"CC"`. In other cases it falls-back to calling `color_of.numeric()`. Returned color definitions always have default names irrespective of names of `x`, which is different from the behavior of `color_of()` methods.

Function `fast_color_of_wb()` accepts waveband objects and lists of waveband objects. If all wavebands are narrow, it issues a vectotized call to `fast_color_of_wl()` with a vector of waveband midpoint wavelengths.

Examples

```
wavelengths <- c(300, 420, 500, 600, NA) # nanometres
color_of(wavelengths)
color_of(waveband(c(300,400)))
color_of(list(blue = waveband(c(400,480)), red = waveband(c(600,700))))
color_of(numeric())
color_of(NA_real_)

color_of(sun.spct)
```

`compare_spct` *Coarse-grained comparison of two spectra*

Description

Compare two spectra using a specified summary function pre-applied to wavelength intervals.

Usage

```
compare_spct(
  x,
  w.band = 10,
  .summary.fun = NULL,
  ...,
  .comparison.fun = `/`,
  returned.value = "spectrum",
  use.hinges = FALSE,
  short.names = TRUE
)
```

Arguments

`x`	A collection of two spectral objects of the same type.
`w.band`	waveband object or a numeric stepsize in nanometres.
`.summary.fun`	function. The summary function to use. It must be a method accepting object `x` as first argument.
`...`	additional named arguments passed down to `.summary.fun`.
`.comparison.fun`	function. The comparison function to use.
`returned.value`	character One of "data.frame", "spectrum", "tagged.spectrum".
`use.hinges`	logical Flag indicating whether to insert "hinges" into the returned spectrum when tagging it.
`short.names`	logical Flag indicating whether to use short or long names for wavebands when tagging.

Details

Summaries are computed for each of the wavebands in `w.band` by applying function `.summary.fun` separately to each spectrum, after trimming them to the overlapping wavelength region. Next the matching summaries are compared by means of `.comparison.fun`. Both the summaries and the result of the comparison are returned. Columns containing summary values are named by concatenating the name each member spectrum with the name of the argument passed to `.summary.fun`.

Tagging is useful for plotting using wavelength based colours, or when names for wavebands are used as annotations. When tagging is requested, the spectrum is passed to method `tag` with `use.hinges` and `short.names` as additional arguments.

Value

A `generic_spct`, tagged or not with the wavebdans, or a `data.frame` object containing the summary values per waveband for each spectrum and the result of applying the comparison function to these summaries.

Examples

```
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)))
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
             w.band = NULL)
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
             w.band = list(waveband(c(640, 650)), waveband(c(720, 740))))

compare_spct(filter_mspct(list(pet = polyester.spct,
                               yllw = yellow_gel.spct)),
             w.band = 50,
             .comparison.fun = `<`)

head(
  compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
               returned.value = "data.frame")
)
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
             returned.value = "tagged.spectrum")
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
             returned.value = "tagged.spectrum",
             use.hinges = TRUE)
```

cone_fundamentals10.spct *Ten-degree cone fundamentals*

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding response values for a 2 degrees target. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

Usage

```
cone_fundamentals10.spct

cone_fundamentals10.mspct
```

Format

A chroma_spct object with 440 rows and 4 variables

An object of class `response_mspct` (inherits from `generic_mspct`, `list`) with 3 rows and 1 columns.

Details

- w.length (nm)
- x
- y
- z

Value

A `chroma_spct` object.

A `response_mspct` object containing the same data in three `response_spct` objects, one for each of x, y and z.

Note

These data are not from the official CIE on-line distribution but are retained for backwards compatibility. It is recommended to download the latest version from https://cie.co.at/data-tables.

The missing data for z in the NIR have been filled with zeros.

Author(s)

CIE

See Also

Other Visual response data examples: beesxyzCMF.spct, ciev10.spct, ciev2.spct, ciexyzCC10.spct, ciexyzCC2.spct, ciexyzCMF10.spct, ciexyzCMF2.spct

Examples

```
cone_fundamentals10.spct
```

---

convertTfrType *Convert the "Tfr.type" attribute*

Description

Function to set the "Tfr.type" attribute and simultaneously convert the spectral data to correspond to the new type.

Usage

```
convertTfrType(x, Tfr.type = NULL)
```

Arguments

x	a filter_spct, object_spct, filter_mspct or object_mspct object.
Tfr.type	character One of "internal" or "total".

Details

Internal transmittance, τ, uses as reference the light entering the object while total transmittance, T, takes the incident light as reference. The conversion is possible only if total reflectance, ρ, is known. Either as spectral data in an object_spct object, a filter_spct object that is "under-the-hood" an object_spct, or if a fixed reflectance factor applicable to all wavelengths is stored in the filter.properties attribute of the filter_spct object.

Conversions are computed as:

$$\tau = \frac{T - \rho}{1 - \rho}$$

and

$$T = \tau * (1 - \rho) + \rho$$

For the conversion to take place the object passed as argument to x, must contain a column with transmittance data, named Tfr. Any necessary conversion from absorbance A or from Afr into transmittance, must be done before calling convertTfrType().

Value

x if possible, with the value of the "Tfr.type" attribute modified and the values stored in the Tfr variable converted to the new quantity.

Note

if x is not a filter_spct object, x is returned unchanged. If x does not have the "filter.properties" attribute set if it is missing data, x is returned with Tfr set to NA values.

See Also

setTfrType, filter_spct

Examples

```
getTfrType(polyester.spct)
filter_properties(polyester.spct)
convertTfrType(polyester.spct, Tfr.type = "internal")
```

`convertThickness` *Convert the "thickness" attribute of an existing filter_spct object.*

Description

Function to set the "thickness" attribute and simultaneously converting the spectral data to correspond to the new thickness.

Usage

```
convertThickness(x, thickness = NULL)
```

Arguments

x	a filter_spct, object_spct, filter_mspct or object_mspct object.
thickness	numeric $[m]$.

Details

For spectral transmittance at a different thickness to be exactly computed, it needs to be based on internal transmittance. This function will apply converTfrType() to x if needed, but to succeed metadata should be available. Please, see convertTfrType.

Value

x possibly with the "thickness" field of the "filter.properties" attribute modified and Tfr or A computed for the requested thickness.

Note

if `x` is not a `filter_spct`, `object_spct`, `filter_mspct` or `object_mspct` object or a collection of such objects, `x` is returned unchanged. If `x` does not have the `"filter.properties"` attribute set or has it with missing member data, `x` is returned with `Tfr` set to `NA` values.

See Also

Other time attribute functions: `checkTimeUnit()`, `convertTimeUnit()`, `getTimeUnit()`, `setTimeUnit()`

Examples

```
my.spct <- polyester.spct
filter_properties(my.spct)
convertThickness(my.spct, thickness = 250e-6)
```

`convertTimeUnit` *Convert the "time.unit" attribute of an existing source_spct object*

Description

Function to set the "time.unit" attribute and simultaneously rescaling the spectral data to be expressed using the new time unit as basis of expression. The change is done by reference ('in place').

Usage

```
convertTimeUnit(x, time.unit = NULL, ...)
```

Arguments

`x`	source_spct or response_spct object
`time.unit`	a character string, either "second", "hour", "day", "exposure" or "none", or a lubridate::duration
`...`	(currently ignored)

Value

x possibly with the `time.unit` attribute modified

Note

if x is not a `source_spct` or a `response_spct` object, or time.unit is NULL x is returned unchanged, if the existing or new time.unit cannot be converted to a duration, then the returned spectrum will contain NAs.

See Also

Other time attribute functions: `checkTimeUnit()`, `convertThickness()`, `getTimeUnit()`, `setTimeUnit()`

Examples

```
my.spct <- sun.spct
my.spct
convertTimeUnit(my.spct, "day")
my.spct
```

---

convolve_each *Convolve function for collections of spectra*

Description

Convolve function for collections of spectra which applies an operation on all the individual members of the collection(s) of spectra.

Usage

```
convolve_each(e1, e2, oper = `*`, sep = "_", ...)
```

Arguments

`e1`	an object of class `generic_mspct` or `generic_scpt` or `numeric`
`e2`	an object of class `generic_mspct` or `generic_scpt` or `numeric`
`oper`	function, usually but not necessarily an operator with two arguments.
`sep`	character Used when pasting the names of members of `e1` and `e2` to form the names of members of the returned collection of spectra.
`...`	additional arguments passed to `oper` if present.

Note

At least one of e1 and e2 must be a `generic_mspct` object or derived.

See Also

Other math operators and functions: `MathFun`, `^.generic_spct()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`

copy_attributes *Copy attributes*

Description

Copy attributes from x to y. Methods defined for spectral and waveband objects of classes from package 'photobiology'.

Usage

```
copy_attributes(x, y, which, ...)

## Default S3 method:
copy_attributes(x, y, which = NULL, ...)

## S3 method for class 'generic_spct'
copy_attributes(x, y, which = NULL, which.not = NULL, copy.class = FALSE, ...)

## S3 method for class 'generic_mspct'
copy_attributes(x, y, which = NULL, which.not = NULL, copy.class = FALSE, ...)

## S3 method for class 'waveband'
copy_attributes(x, y, which = NULL, ...)
```

Arguments

x, y	R objects
which	character Names of attributes to copy, if NULL all those relevant according to the class of x is used as defaul,
...	not used
which.not	character Names of attributes not to be copied. The names passed here are removed from the list for which, which is most useful when we want to modify the default.
copy.class	logical If TRUE class attributes are also copied.

Value

A copy of y with additional attributes set.

Methods (by class)

- `copy_attributes(default)`: Default for generic function
- `copy_attributes(generic_spct)`:
- `copy_attributes(generic_mspct)`:
- `copy_attributes(waveband)`:

`cps2irrad` *Conversion from counts per second to physical quantities*

Description

Conversion of spectral data expressed as cps into irradiance, transmittance or reflectance.

Usage

```
cps2irrad(x.sample, pre.fun = NULL, missing.pixs = numeric(0), ...)

cps2Rfr(x.sample, x.white, x.black = NULL, dyn.range = NULL)

cps2Tfr(x.sample, x.clear, x.opaque = NULL, dyn.range = NULL)
```

Arguments

`x.sample`, `x.clear`, `x.opaque`, `x.white`, `x.black`	cps_spct objects.
`pre.fun`	function A function applied to x.sample before conversion.
`missing.pixs`	integer Index to positions in the detector array or scan missing in `x.sample` but present in the embedded calibration data. (Use only for emergency recovery of incomplete data!!)
`...`	Additional arguments passed to `pre.fun`.
`dyn.range`	numeric The effective dynamic range of the instrument, if `NULL` it is automatically set based on integration time bracketing.

Value

A source_spct, filter_spct or reflector_spct object containing the spectral values expressed in physical units.

Note

In contrast to other classes defined in package 'photobiology', class "cps_spct" can have more than one column of cps counts in cases where the intention is to merge these values as part of the processing at the time the calibration is applied. However, being these functions the final step in the conversion to physical units, they accept as input only objects with a single "cps" column, as merging is expected to have been already done.

D2.UV653 *Data for typical calibration lamps*

Description

A dataset containing fitted constants to be used as input for functions D2_spectrum and FEL_spectrum for computing example spectral curves based on fitted polynomials.

Format

A `polynom::polynomial` object with 6 constants.

Details

An object of class `polynom::polynomial`.

Author(s)

Lasse Ylianttila (data)

Examples

```
D2.UV653
as.character(D2.UV653)
```

D2_spectrum *Calculate deuterium lamp output spectrum from fitted constants*

Description

Calculate values by means of a nth degree polynomial from user-supplied constants (for example from a lamp calibration certificate).

Usage

```
D2_spectrum(w.length, k = photobiology::D2.UV653, fill = NA_real_)
```

Arguments

`w.length`	numeric vector of wavelengths (nm) for output
`k`	a polynom:polynomial object with n constants for the polynomial
`fill`	if NA, no extrapolation is done, and NA is returned for wavelengths outside the range 190 nm to 450 nm. If NULL then the tails are deleted. If 0 then the tails are set to zero, etc. NA is default.

Value

a dataframe with four numeric vectors with wavelength values (w.length), energy and photon irradiance (s.e.irrad, s.q.irrad) depending on the argument passed to unit.out (s.irrad).

Note

This is function is valid for wavelengths in the range 180 nm to 495 nm, for wavelengths outside this range NAs are returned.

Examples

```
D2_spectrum(200)
D2_spectrum(170:220)
```

`D50.illuminant.spct` *CIE D50 illuminant data*

Description

A dataset containing wavelengths at a 5 nm interval (300 nm to 830 nm) and the corresponding spectral energy irradiance normalized to 1 at 560 nm. Spectrum approximates the midday solar spectrum at middle latitude as 'corresponds' to the white point of a black body a 6504 K. Original data from CIE downloaded on 2024-11-30 The variables are as follows:

Usage

```
D50.illuminant.spct
```

Format

A source spectrum with 531 rows and 2 variables

- w.length (nm)
- s.e.irrad (rel. units)

Note

This and other CIE illuminant spectra can be downloaded from `https://cie.co.at/data-tables` as .CSV files.

Author(s)

CIE

References

CIE 2022, Relative spectral power distributions of CIE standard illuminants A, D65 and D50 (wavelengths in standard air) (data table), International Commission on Illumination (CIE), Vienna, Austria, doi:10.25039/CIE.DS.etgmuqt5.

See Also

Other Spectral data examples: `A.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
D50.illuminant.spct
```

`D65.illuminant.spct` *CIE D65 illuminant data*

Description

A dataset containing wavelengths at a 5 nm interval (300 nm to 830 nm) and the corresponding spectral energy irradiance normalized to 1 at 560 nm. Spectrum approximates the midday solar spectrum at middle latitude as 'corresponds' to the white point of a black body a 6504 K. Original data from CIE downloaded on 2024-11-30 The variables are as follows:

Usage

```
D65.illuminant.spct
```

Format

A source spectrum with 531 rows and 2 variables

- w.length (nm)
- s.e.irrad (rel. units)

Note

This and other CIE illuminant spectra can be downloaded from `https://cie.co.at/data-tables` as .CSV files.

Author(s)

CIE

References

CIE 2022, CIE standard illuminant D65, International Commission on Illumination (CIE), Vienna, Austria, doi:10.25039/CIE.DS.hjfjmt59.

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
D65.illuminant.spct
```

`defunct` *Defunct functions and methods*

Description

Functions listed here have been removed or deleted, and temporarily replaced by stubs that report this when they are called.

Usage

```
f_mspct(...)

mutate_mspct(...)

calc_filter_multipliers(...)

T2T(...)

getAfrType(...)

setAfrType(...)

sample_spct(...)

sample_mspct(...)
```

Arguments

`...`	ignored

Note

Function `f_mspct()` has been renamed `msdply()`.

Function `mutate_mspct()` has been renamed `msmsply()`.

Function `calc_filter_multipliers()` has been removed.

Function `calc_filter_multipliers()` has been removed.

Method `getAfrType()` has been removed.

Method `setAfrType()` has been removed.

Function `sample_spct()` has been removed.

Function `sample_mspct()` has been removed.

`despike` *Remove spikes from spectrum*

Description

Function that returns an R object with observations corresponding to spikes replaced by values computed from neighboring pixels. Spikes are values in spectra that are unusually high compared to neighbors. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode array detectors.

Usage

```
despike(x, z.threshold, max.spike.width, window.width, method, na.rm, ...)

## Default S3 method:
despike(
  x,
  z.threshold = NA,
  max.spike.width = NA,
  window.width = NA,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'numeric'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'data.frame'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  y.var.name = NULL,
  var.name = y.var.name
)

## S3 method for class 'generic_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  y.var.name = NULL,
  var.name = y.var.name,
  ...
)

## S3 method for class 'source_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'solute_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
```

```
)

## S3 method for class 'raw_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'generic_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  y.var.name = NULL,
  var.name = y.var.name,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'source_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

```
## S3 method for class 'cps_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object
`z.threshold`	numeric Modified Z values larger than `z.threshold` are considered to correspond to spikes.
`max.spike.width`	integer Wider regions with high Z values are not detected as spikes.
`window.width`	integer. The full width of the window used for the running mean used as replacement.
`method`	character The name of the method: `"run.mean"` is running mean as described in Whitaker and Hayes (2018); `"adj.mean"` is mean of adjacent neighbors (isolated bad pixels only).
`na.rm`	logical indicating whether `NA` values should be treated as spikes and replaced.
`...`	Arguments passed by name to `find_spikes()`.
`var.name`, `y.var.name`	character Names of columns where to look for spikes to remove.
`unit.out`	character One of "energy" or "photon"
`filter.qty`	character One of "transmittance" or "absorbance"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach

`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, resulting in most cases in more spikes being detected. A modified version of the algorithm is used if a value different from `NULL` is passed as argument to `max.spike.width`. In such a case, an additional step filters out broader spikes (or falsely detected steep slopes) from the returned values.

Simple interpolation replaces values of isolated bad pixels by the mean of their two closest neighbors. The running mean approach allows the replacement of short runs of bad pixels by the running mean of neighboring pixels within a window of user-specified width. The first approach works well for spectra from array spectrometers to correct for hot and dead pixels in an instrument. The second approach is most suitable for Raman spectra in which spikes triggered by radiation are wider than a single pixel but usually not more than five pixels wide.

When the argument passed to `x` contains multiple spectra, the spikes are searched for and replaced in each spectrum independently of other spectra.

Value

A copy of the object passed as argument to `x` with values detected as spikes replaced by a local average of adjacent neighbors outside the spike.

Methods (by class)

- `despike(default)`: Default returning always NA.
- `despike(numeric)`: Default function usable on numeric vectors.
- `despike(data.frame)`: Method for "data.frame" objects.
- `despike(generic_spct)`: Method for "generic_spct" objects.
- `despike(source_spct)`: Method for "source_spct" objects.
- `despike(response_spct)`: Method for "response_spct" objects.
- `despike(filter_spct)`: Method for "filter_spct" objects.
- `despike(reflector_spct)`: Method for "reflector_spct" objects.
- `despike(solute_spct)`: Method for "solute_spct" objects.
- `despike(cps_spct)`: Method for "cps_spct" objects.
- `despike(raw_spct)`: Method for "raw_spct" objects.
- `despike(generic_mspct)`: Method for "generic_mspct" objects.
- `despike(source_mspct)`: Method for "source_mspct" objects.
- `despike(response_mspct)`: Method for "cps_mspct" objects.
- `despike(filter_mspct)`: Method for "filter_mspct" objects.

- `despike(reflector_mspct)`: Method for "reflector_mspct" objects.
- `despike(solute_mspct)`: Method for "solute_mspct" objects.
- `despike(cps_mspct)`: Method for "cps_mspct" objects.
- `despike(raw_mspct)`: Method for "raw_mspct" objects.

Note

Current algorithm misidentifies steep smooth slopes as spikes, so manual inspection is needed together with adjustment by trial and error of a suitable argument value for `z.threshold`.

See Also

See the documentation for `find_spikes` and `replace_bad_pixs` for details of the algorithm and implementation.

Examples

```
white_led.raw_spct[120:125, ]

# find and replace spike at 245.93 nm
despike(white_led.raw_spct,
        z.threshold = 10,
        window.width = 25)[120:125, ]
```

`diffraction_single_slit` *Diffraction*

Description

Diffraction of optical radiation passing through a single slit can be computed with function `diffraction_single_slit()`, which implements Fraunhofer's equation. Diffraction plus interference for a pair of slits can be computed with `diffraction_double_slit()`.

Usage

```
diffraction_single_slit(w.length, slit.width, angle)

diffraction_double_slit(w.length, slit.width, slit.distance, angle)
```

Arguments

`w.length`	numeric Wavelength (nm).
`slit.width`	numeric Width of the slit (m).
`angle`	numeric vector Angle (radians).
`slit.distance`	numeric Distance between the centres of the two slits (m).

Value

A numeric vector of the same length as `angle`, containing relative intensities.

Examples

```
diffraction_single_slit(w.length = 550,
                        slit.width = 1e-5,
                        angle = 0)

# use odd number for length.out so that 0 is in the sequence
angles <- pi * seq(from = -1/2, to = 1/2, length.out = 501)

plot(angles,
     diffraction_single_slit(w.length = 550, # 550 nm
                        slit.width = 6e-6, # 6 um
                        angle = angles),
     type = "l",
     ylab = "Relative irradiance (/1)",
     xlab = "Angle (radian)")

plot(angles,
     diffraction_double_slit(w.length = 550, # 550 nm
                        slit.width = 6e-6, # 6 um
                        slit.distance = 18e-6, # 18 um
                        angle = angles),
     type = "l",
     ylab = "Relative irradiance (/1)",
     xlab = "Angle (radian)")
```

`dim.generic_mspct` *Dimensions of an Object*

Description

Retrieve or set the dimension of an object.

Usage

```
## S3 method for class 'generic_mspct'
dim(x)

## S3 replacement method for class 'generic_mspct'
dim(x) <- value
```

Arguments

`x`	A `generic_mspct` object or of a derived class.
`value`	Either NULL or a numeric vector, which is coerced to integer (by truncation).

Value

Either NULL or a numeric vector, which is coerced to integer (by truncation).

---

div-.generic_spct *Arithmetic Operators*

Description

Integer-division operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 %/% e2
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

---

div_spectra *Divide two spectra, even if the wavelengths values differ*

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are operated upon.

Usage

```
div_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

Arguments

`w.length1`	numeric vector of wavelength (nm) of denominator.
`w.length2`	numeric vector of wavelength (nm) of divisor.
`s.irrad1`	a numeric vector of spectral values of denominator.
`s.irrad2`	a numeric vector of spectral values of divisor.
`trim`	a character string with value "union" or "intersection".
`na.rm`	a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a dataframe with two numeric variables.

`w.length`	A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique vales, sorted in ascending order.
`s.irrad`	A numeric vector with the ratio between the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
head(sun.data)
one.data <-
  with(sun.data, div_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(one.data)
tail(one.data)
```

drop_user_cols *Drop user columns*

Description

Remove from spectral object additional columns that are user defined.

Usage

```
drop_user_cols(x, keep.also, ...)

## Default S3 method:
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'generic_spct'
drop_user_cols(x, keep.also, ...)

## S3 method for class 'source_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'response_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'object_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'filter_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'reflector_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'solute_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'chroma_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'calibration_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'cps_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'raw_spct'
drop_user_cols(x, keep.also = NULL, ...)
```

```
## S3 method for class 'generic_mspct'
drop_user_cols(x, keep.also = NULL, ...)
```

Arguments

`x`	An R object
`keep.also`	character Additionlal columns to preserve.
`...`	needed to allow derivation.

Value

A copy of x possibly with some columns removed.

Methods (by class)

- `drop_user_cols(default)`:
- `drop_user_cols(generic_spct)`:
- `drop_user_cols(source_spct)`:
- `drop_user_cols(response_spct)`:
- `drop_user_cols(object_spct)`:
- `drop_user_cols(filter_spct)`:
- `drop_user_cols(reflector_spct)`:
- `drop_user_cols(solute_spct)`:
- `drop_user_cols(chroma_spct)`:
- `drop_user_cols(calibration_spct)`:
- `drop_user_cols(cps_spct)`:
- `drop_user_cols(raw_spct)`:
- `drop_user_cols(generic_mspct)`:

See Also

Other experimental utility functions: `collect2mspct()`, `thin_wl()`, `uncollect2spct()`

`e2q` *Convert energy-based quantities into photon-based quantities.*

Description

Conversion methods for spectral energy irradiance into spectral photon irradiance and for spectral energy response into spectral photon response.

Usage

```
e2q(x, action, byref, ...)

## Default S3 method:
e2q(x, action = "add", byref = FALSE, ...)

## S3 method for class 'source_spct'
e2q(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'response_spct'
e2q(x, action = "add", byref = FALSE, ...)

## S3 method for class 'source_mspct'
e2q(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)

## S3 method for class 'response_mspct'
e2q(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
```

Arguments

`x`	an R object.
`action`	a character string, one of "add", or "replace".
`byref`	logical indicating if a new object will be created by reference or a new object returned.
`...`	not used in current version.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

The converted spectral values are added to or replace the existing spectral values depending on the argument passed to parameter `action`. Addition is currently not supported for normalized spectra. If the spectrum has been normalized with a recent version of package 'photobiology' the spectrum will be renormalized after conversion using the same arguments as previously.

Methods (by class)

- `e2q(default)`: Default method
- `e2q(source_spct)`: Method for spectral irradiance
- `e2q(response_spct)`: Method for spectral responsiveness
- `e2q(source_mspct)`: Method for collections of (light) source spectra
- `e2q(response_mspct)`: Method for collections of response spectra

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

`e2qmol_multipliers` *Calculate energy to quantum (mol) multipliers*

Description

Multipliers as a function of wavelength, for converting from energy to photon (quantum) molar units.

Usage

```
e2qmol_multipliers(w.length)
```

Arguments

`w.length`	numeric Vector of wavelengths (nm)

Value

A numeric vector of multipliers

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2quantum_multipliers()`, `q2e()`

Examples

```
with(sun.data, e2qmol_multipliers(w.length))
```

`e2quantum_multipliers` *Calculate energy to quantum multipliers*

Description

Gives multipliers as a function of wavelength, for converting from energy to photon (quantum) units (number of photons as default, or moles of photons).

Usage

```
e2quantum_multipliers(w.length, molar = FALSE)
```

Arguments

`w.length`	numeric Vector of wavelengths (nm)
`molar`	logical Flag indicating whether output should be in moles or numbers

Value

A numeric vector of multipliers

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `q2e()`

Examples

```
with(sun.data, e2quantum_multipliers(w.length))
with(sun.data, e2quantum_multipliers(w.length, molar = TRUE))
```

`enable_check_spct` *Enable or disable checks*

Description

Choose between protection against errors or faster performance by enabling (the default) or disabling data-consistency and sanity checks.

Usage

```
enable_check_spct()

disable_check_spct()

set_check_spct(x)
```

Arguments

`x`	logical Flag to enable (TRUE), disable (FALSE) or unset (NULL) option.

Details

Checks are applied by default after each operation that modifies the data. This can be excessive in production code. Some functions within this package disable checks for partial computations and apply them to the value they return. It is possible for users to apply this same approach, in which case it is best to schedule the restore of the previous setting using ‘on.exit()‘.

Value

The previous value of the option, which can be passed as argument to function `set_check_spct()` to restore the previous state of the option.

See Also

[check_spct()]

Other data validity check functions: `check_spct()`, `check_spectrum()`, `check_w.length()`

`energy_as_default` *Set spectral-data options*

Description

Set spectral-data related options easily.

Usage

```
energy_as_default()

photon_as_default()

quantum_as_default()

Tfr_as_default()

Afr_as_default()

A_as_default()

unset_radiation_unit_default()

unset_filter_qty_default()

unset_user_defaults()
```

Value

Previous value of the modified option.

energy_irradiance *Calculate (energy) irradiance from spectral irradiance*

Description

Energy irradiance for a waveband from a radiation spectrum, optionally applying a "biological spectral weighting function" or BSWF.

Usage

```
energy_irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelength $[nm]$.
`s.irrad`	numeric vector of spectral irradiances in $[W\,m^{-2}\,nm^{-1}]$ or $[mol\,s^{-1}\,sm^{-2}\,nm^{-1}]$ as indicated by the argument pased to `unit.in`.
`w.band`	waveband.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is `TRUE`.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value with no change in scale factor: $[W\,m^{-2}]$.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data, energy_irradiance(w.length, s.e.irrad))
with(sun.data, energy_irradiance(w.length, s.e.irrad, new_waveband(400,700)))
```

`energy_ratio` *Energy:energy ratio*

Description

Energy irradiance ratio between two wavebands for a radiation spectrum.

Usage

```
energy_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = NULL
)
```

Arguments

`w.length`	numeric vector of wavelengths [nm].
`s.irrad`	numeric vector of spectral irradiances in [$W\,m^{-2}\,nm^{-1}$] or [$mol\,s^{-1}\,sm^{-2}\,nm^{-1}$] as indicated by the argument pased to `unit.in`.
`w.band.num`	waveband object used to compute the numerator of the ratio.
`w.band.denom`	waveband object used to compute the denominator of the ratio.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is `TRUE`.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a single numeric value giving the unitless energy ratio.

Note

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data,
     energy_ratio(w.length, s.e.irrad,
                  new_waveband(400,500), new_waveband(400,700)))
```

`eq_ratio` *Energy:photon ratio*

Description

This function returns the energy to mole of photons ratio for each waveband and a light source spectrum.

Usage

```
eq_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## Default S3 method:
eq_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## S3 method for class 'source_spct'
eq_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = "[e:q]",
  ...
)
```

```
## S3 method for class 'source_mspct'
eq_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = "[e:q]",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	source_spct.
`w.band`	waveband or list of waveband objects.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

The ratio is based on one photon irradiance and one energy irradiance, both computed for the same waveband.

$$\frac{I(s, wb)}{Q(s, wb)}$$

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.#' @return Computed values are ratios between energy irradiance and photon irradiance for a given waveband. A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used, with "[e:q]" prepended. Units [J mol-1].

Value

Computed values are ratios between energy irradiance and photon irradiance for a given waveband. A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of multiple spectra, containing one column with ratios for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they are expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used, with "[e:q]" prepended. Units [mol J-1].

Performance

As this method accepts spectra as its input, it computes irradiances before computing the ratios. If you need to compute both ratios and irradiances from several hundreds or thousands of spectra, computing the ratios from previously computed irradiances avoids their repeated computation. A less dramatic, but still important, increase in performance is available when computing in the same function call ratios that share the same denominator.

See Also

Other photon and energy ratio functions: `e_fraction()`, `e_ratio()`, `q_fraction()`, `q_ratio()`, `qe_ratio()`

Examples

```
eq_ratio(sun.spct,
         waveband(c(400,700), wb.name = "White")) # J mol-1
eq_ratio(sun.spct,
         waveband(c(400,700), wb.name = "White"),
         scale.factor = 1e-6) # J umol-1
```

Extract *Extract or replace parts of a spectrum*

Description

Just like extraction and replacement with indexes in base R, but preserving the special attributes used in spectral classes and checking for validity of remaining spectral data.

Usage

```
## S3 method for class 'generic_spct'
x[i, j, drop = NULL]

## S3 method for class 'raw_spct'
x[i, j, drop = NULL]

## S3 method for class 'cps_spct'
x[i, j, drop = NULL]

## S3 method for class 'source_spct'
x[i, j, drop = NULL]

## S3 method for class 'response_spct'
x[i, j, drop = NULL]

## S3 method for class 'filter_spct'
x[i, j, drop = NULL]

## S3 method for class 'reflector_spct'
x[i, j, drop = NULL]

## S3 method for class 'solute_spct'
x[i, j, drop = NULL]

## S3 method for class 'object_spct'
x[i, j, drop = NULL]

## S3 method for class 'chroma_spct'
x[i, j, drop = NULL]
```

```
## S3 replacement method for class 'generic_spct'
x[i, j] <- value

## S3 replacement method for class 'generic_spct'
x$name <- value
```

Arguments

`x`	spectral object from which to extract element(s) or in which to replace element(s)
`i`	index for rows,
`j`	index for columns, specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or NULL. Please, see `Extract` for more details.
`drop`	logical. If TRUE the result is coerced to the lowest possible dimension. The default is FALSE unless the result is a single column.
`value`	A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.
`name`	A literal character string or a name (possibly backtick quoted). For extraction, this is normally (see under 'Environments') partially matched to the names of the object.

Details

These methods are just wrappers on the method for data.frame objects which copy the additional attributes used by these classes, and validate the extracted object as a spectral object. When drop is TRUE and the returned object has only one column, then a vector is returned. If the extracted columns are more than one but do not include `w.length`, a data frame is returned instead of a spectral object.

Value

An object of the same class as `x` but containing only the subset of rows and columns that are selected. See details for special cases.

Note

If any argument is passed to `j`, even `TRUE`, some metadata attributes are removed from the returned object. This is how the extraction operator works with `data.frames` in R. For the time being we retain this behaviour for spectra, but it may change in the future.

See Also

`subset` and `trim_spct`

Examples

```
sun.spct[sun.spct[["w.length"]] > 400, ]
subset(sun.spct, w.length > 400)

tmp.spct <- sun.spct
tmp.spct[tmp.spct[["s.e.irrad"]] < 1e-5 , "s.e.irrad"] <- 0
e2q(tmp.spct[ , c("w.length", "s.e.irrad")]) # restore data consistency!
```

`Extract_mspct` *Extract or replace members of a collection of spectra*

Description

Just like extraction and replacement with indexes for base R lists, but preserving the special attributes used in spectral classes.

Usage

```
## S3 method for class 'generic_mspct'
x[i, drop = NULL]

## S3 replacement method for class 'generic_mspct'
x[i] <- value

## S3 replacement method for class 'generic_mspct'
x$name <- value

## S3 replacement method for class 'generic_mspct'
x[[name]] <- value
```

Arguments

`x`	Collection of spectra object from which to extract member(s) or in which to replace member(s)
`i`	Index specifying elements to extract or replace. Indices are numeric or character vectors. Please, see `Extract` for more details.
`drop`	If TRUE the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement.
`value`	A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.
`name`	A literal character string or a name (possibly backtick quoted). For extraction, this is normally (see under 'Environments') partially matched to the names of the object.

Details

This method is a wrapper on base R's extract method for lists that sets additional attributes used by these classes.

Value

An object of the same class as x but containing only the subset of members that are selected.

---

e_fluence *Energy fluence*

Description

Energy fluence for one or more wavebands of a light source spectrum and a duration of the exposure.

Usage

```
e_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
e_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
e_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
e_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object
`w.band`	a list of `waveband` objects or a `waveband` object
`exposure.time`	lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error
`...`	other arguments (possibly ignored)
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The exposure.time is copied to the output as an attribute. Units are as follows: (J) joules per exposure.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_irrad()`, `fluence()`, `irrad()`, `q_fluence()`, `q_irrad()`

Examples

```
library(lubridate)
e_fluence(sun.spct, w.band = waveband(c(400,700)),
          exposure.time = lubridate::duration(3, "minutes") )
```

`e_fraction` *Energy:energy fraction*

Description

This function returns the energy fraction for a given pair of wavebands of a light source spectrum.

Usage

```
e_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
e_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'source_spct'
e_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'source_mspct'
e_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = ifelse(naming != "none", "[e:e]", ""),
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	source_spct
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the fraction(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the fraction(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach.
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "total"` the fraction is based on two **energy irradiances**, one computed for each waveband.

$$\frac{E(s, wb_{\mathrm{num}})}{E(s, wb_{\mathrm{denom}}) + E(s, wb_{\mathrm{num}})}$$

If the argument is set to `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral energy irradiances**, one computed for each waveband.

$$\frac{\overline{Q_\lambda}(s, wb_{\mathrm{num}})}{\overline{Q_\lambda}(s, wb_{\mathrm{denom}}) + \overline{Q_\lambda}(s, wb_{\mathrm{num}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[e:e]" is appended if `quantity = "total"` and "[e(wl):e(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `e_fraction(default)`: Default for generic function
- `e_fraction(source_spct)`: Method for `source_spct` objects
- `e_fraction(source_mspct)`: Calculates energy:energy fraction from a `source_mspct` object.

Note

Recycling for wavebands takes place when the number of denominator and denominator wavebands differ. The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other photon and energy ratio functions: `e_ratio()`, `eq_ratio()`, `q_fraction()`, `q_ratio()`, `qe_ratio()`

Examples

```
e_fraction(sun.spct, new_waveband(400,700), new_waveband(400,500))
```

`e_irrad` *Energy irradiance*

Description

Energy irradiance for one or more wavebands of a light source spectrum.

Usage

```
e_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
e_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  return.tb = FALSE,
  ...
)

## S3 method for class 'source_mspct'
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	a list of `waveband` objects or a `waveband` object.
`quantity`	character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
`time.unit`	character or lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.

`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`return.tb`	logical Flag forcing a tibble to be always returned, even for a single spectrum as argumnet to `spct`. The default is `FALSE` for backwards compatibility.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of a `_spct` object containing a single spectrum and `return.tb = FALSE`. The vector has one member one value for each `waveband` passed to parameter `w.band`. In all other cases a `tibble`, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If units are absolute and time.unit is second, [W m-2 nm-1] -> [W m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [J m-2]; if units are relative, fraction of one or percent.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_fluence()`, `fluence()`, `irrad()`, `q_fluence()`, `q_irrad()`

Examples

```
e_irrad(sun.spct, waveband(c(400,700)))
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3))
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "total")
```

```
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "average")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "relative.pc")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
        quantity = "contribution.pc")
```

---

e_ratio *Energy:energy ratio*

Description

This function returns the photon ratio for a given pair of wavebands of a light source spectrum.

Usage

```
e_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
e_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'source_spct'
e_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'source_mspct'
e_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = "[e:e]",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	source_spct
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) of the ratio(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).

`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach.
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "total"` the ratio is based on two energy irradiances, one computed for each waveband.

$$\frac{I(s, wb_{\mathrm{num}})}{I(s, wb_{\mathrm{denom}})}$$

If the argument is set to `quantity = "mean"` or `quantity = "average"` the ratio is based on two mean spectral photon irradiances, one computed for each waveband.

$$\frac{\overline{I_\lambda}(s, wb_{\mathrm{num}})}{\overline{I_\lambda}(s, wb_{\mathrm{denom}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[e:e]" is appended if `quantity = "total"` and "[e(wl):e(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Performance

As this method accepts spectra as its input, it computes irradiances before computing the ratios. If you need to compute both ratios and irradiances from several hundreds or thousands of spectra, computing the ratios from previously computed irradiances avoids their repeated computation. A less dramatic, but still important, increase in performance is available when computing in the same function call ratios that share the same denominator.

See Also

Other photon and energy ratio functions: `e_fraction()`, `eq_ratio()`, `q_fraction()`, `q_ratio()`, `qe_ratio()`

Examples

```
e_ratio(sun.spct,
        waveband(c(400,500), wb.name = "Blue"),
        waveband(c(400,700), wb.name = "White"))
```

`e_response` *Energy-based photo-response*

Description

This function returns the mean, total, or contribution of response for each waveband and a response spectrum.

Usage

```
e_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## Default S3 method:
e_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## S3 method for class 'response_spct'
e_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

## S3 method for class 'response_mspct'
e_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`quantity`	character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
`time.unit`	character or lubridate::duration object.

`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `e_response(default)`: Default method for generic function
- `e_response(response_spct)`: Method for response spectra.
- `e_response(response_mspct)`: Calculates energy response from a `response_mspct`

Note

The parameter `use.hinges` controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also

Other response functions: `q_response()`, `response()`

Examples

```
e_response(ccd.spct, new_waveband(200,300))
e_response(photodiode.spct)
```

FEL_spectrum *Incandescent "FEL" lamp emission spectrum*

Description

Calculate values by means of a nth degree polynomial from user-supplied constants (for example from a lamp calibration certificate).

Usage

```
FEL_spectrum(w.length, k = photobiology::FEL.BN.9101.165, fill = NA_real_)
```

Arguments

`w.length`	numeric vector of wavelengths (nm) for output
`k`	a numeric vector with n constants for the function
`fill`	if NA, no extrapolation is done, and NA is returned for wavelengths outside the range 250 nm to 900 nm. If NULL then the tails are deleted. If 0 then the tails are set to zero, etc. NA is default.

Value

a dataframe with four numeric vectors with wavelength values (w.length), energy and photon irradiance (s.e.irrad, s.q.irrad) depending on the argument passed to unit.out (s.irrad).

Note

This is function is valid for wavelengths in the range 250 nm to 900 nm, for wavelengths outside this range NAs are returned.

Examples

```
FEL_spectrum(400)
FEL_spectrum(250:900)
```

findMultipleWl *Find repeated w.length values*

Description

Find repeated w.length values

Usage

```
findMultipleWl(x, same.wls = TRUE)
```

Arguments

x	a generic_spct object
same.wls	logical If TRUE all spectra spected to share same w.length values.

Value

integer Number of spectra, guessed from the number of copies of each individual w.length value.

find_peaks *Find local maxima or global maximum (peaks)*

Description

These functions find peaks (local maxima) and valleys (local minima) in a numeric vector, using a user selectable span or window. Global and local size thresholds based on different criteria make it possible restrict the returned peaks to those more prominent. A `logical` vector is returned.

Usage

```
find_peaks(
  x,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  span = 3,
  strict = FALSE,
  na.rm = FALSE
)

find_valleys(
  x,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  span = 3,
  strict = FALSE,
  na.rm = FALSE
)
```

Arguments

`x`	numeric vector.
`global.threshold`	numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height or depth expressed in data units. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as relative to `threshold.range`. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are ignored. A bare negative `numeric` value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If `global.threshold = NULL`, no threshold is applied and all peaks returned.
`local.threshold`	numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height (depth) expressed in data units relative to a within-window computed reference value. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as expressed in units relative to `threshold.range`. In both cases `local.threshold` sets a *local* height (depth) threshold below which peaks (valleys) are ignored. If `local.threshold = NULL` or if `span` spans the whole of `x`, no threshold is applied.
`local.reference`	character One of `"median"`, `"median.log"`, `"median.sqrt"`, `"farthest"`, `"farthest.log"` or `"farthest.sqrt"`. The reference used to assess the height of the peak, either the minimum/maximum value within the window or the median of all values in the window.
`threshold.range`	numeric vector If of length 2 or a longer vector `range(threshold.range)` is used to scale both thresholds. With `NULL`, the default, `range(x)` is used, and with a vector of length one `range(threshold.range, x)` is used, i.e., the range is expanded.
`span`	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width `span` centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. `span = NULL` extends the span to the whole length of `x`.
`strict`	logical flag: if `TRUE`, an element must be strictly greater than all other values in its window to be considered a peak.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for peaks.

Details

As find_valleys, peaks and valleys call find_peaks to search for peaks and valleys, this explanation applies to the four functions. It also applies to stat_peaks and stat_valleys. Function find_peaks is a wrapper built onto function peaks from **splus2R**, adds support for peak height thresholds and handles span = NULL and non-finite (including NA) values differently than splus2R::peaks. Instead of giving an error when na.rm = FALSE and x contains NA values, NA values are replaced with the smallest finite value in x. span = NULL is treated as a special case and selects max(x). Passing strict = TRUE ensures that non-unique global and within window maxima are ignored, and can result in no peaks being returned.

Two tests make it possible to ignore irrelevant peaks. One test (global.threshold) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (local.threshold) is available when the window defined by 'span' does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by the argument passed to local.reference. Parameter threshold.range determines how the values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being returned.

The local.threshold argument is used *as is* when local.reference is "median" or "farthest", i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a local.threshold that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with local.reference arguments "median.log", "farthest.log", "median.sqrt", and "farthest.sqrt" as arguments for local.reference.

While functions find_peaks and find_valleys accept as input a numeric vector and return a logical vector, methods peaks and valleys accept as input different R objects, including spectra and collections of spectra and return a subset of the object. These methods are implemented using calls to functions find_peaks, find_valleys and fit_peaks.

Value

A vector of logical values of the same length as x. Values that are TRUE correspond to local peaks in vector x and can be used to extract the rows corresponding to peaks from a data frame.

Note

The default for parameter strict is FALSE in functions find_peaks and find_valleys, while the default in peaks is strict = TRUE.

See Also

peaks.

Other peaks and valleys functions: find_spikes(), get_peaks(), peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples

```
with(sun.data, which(find_peaks(s.e.irrad, span = NULL)))
with(sun.data, which(find_peaks(s.e.irrad, span = 51)))
with(sun.data, w.length[find_peaks(s.e.irrad, span = 51)])
with(sun.data, sum(find_peaks(s.e.irrad, span = NULL, strict = TRUE)))

with(sun.data, which(find_valleys(s.e.irrad, span = NULL)))
with(sun.data, which(find_valleys(s.e.irrad, span = 51)))
```

`find_spikes` *Find spikes*

Description

This function finds spikes in a numeric vector using the algorithm of Whitaker and Hayes (2018). Spikes are values in spectra that are unusually high or low compared to neighbours. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode arrays. Other kinds of accidental "outliers" will be also detected.

Usage

```
find_spikes(
  x,
  x.is.delta = FALSE,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE
)
```

Arguments

`x`	numeric vector containing spectral data.
`x.is.delta`	logical Flag indicating if x contains already differences.
`z.threshold`	numeric Modified Z values larger than `z.threshold` are considered to be spikes.
`max.spike.width`	integer Wider regions with high Z values are not detected as spikes.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for spikes.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, resulting in most cases in more spikes being detected. A modified version of the algorithm is used if a value different from `NULL` is passed as argument to `max.spike.width`. In such a case, an additional step filters out broader spikes (or falsely detected steep slopes) from the returned values.

Value

A logical vector of the same length as `x`. Values that are TRUE correspond to local spikes in the data.

References

Whitaker, D. A.; Hayes, K. (2018) A simple algorithm for despiking Raman spectra. Chemometrics and Intelligent Laboratory Systems, 179, 82-84.

See Also

Other peaks and valleys functions: `find_peaks()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `valleys()`, `wls_at_target()`

Examples

```
with(white_led.raw_spct,
     which(find_spikes(counts_3, z.threshold = 30)))
```

`find_wls` *Find wavelength values in a spectrum*

Description

Find wavelength values corresponding to a target y value in any spectrum. The name of the column of the spectral data to be used to match the target needs to be passed as argument unless the spectrum contains a single numerical variable in addition to "w.length".

Usage

```
find_wls(
  x,
  target = NULL,
  col.name.x = NULL,
  col.name = NULL,
  .fun = `<=`,
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE
)
```

Arguments

x	an R object
target	numeric or character. A numeric value indicates the spectral quantity value for which wavelengths are to be searched. A character representing a number is converted to a number. A character value representing a number followed by a function name, will be also accepted and decoded, such that `"0.1max"` is interpreted as targetting one tenthof the maximum value in a column. The character strings "half.maximum" and "HM" are synonyms for "0.5max" while "half.range" and "HR" are synonyms for "0.5range". These synonyms are converted to the canonical form before saving them to the returned value.
col.name.x	character The name of the column in which to the independent variable is stored. Defaults to "w.length" for objects of class `"generic_spct"` or derived.
col.name	character The name of the column in which to search for the target value.
.fun	function A binary comparison function or operator.
interpolate	logical Indicating whether the nearest wavelength value in `x` should be returned or a value calculated by linear interpolation between wavelength values stradling the target.
idfactor	logical or character Generates an index column of factor type. If `idfactor = TRUE` then the column is auto named target.idx. Alternatively the column name can be directly passed as argument to `idfactor` as a character string.
na.rm	logical indicating whether `NA` values should be stripped before searching for the target.

Value

A spectrum object of the same class as `x` with fewer rows, possibly even no rows. If `FALSE` is passed to `interpolate` a subset of `x` is returned, otherwise a new object of the same class containing interpolated wavelenths for the `target` value is returned.

Note

This function is used internally by method `wls_at_target()`, and these methods should be preferred in user code and scripts.

Examples

```
find_wls(white_led.source_spct)
find_wls(white_led.source_spct, target = "0.5max")
find_wls(white_led.source_spct, target = 0.4)
find_wls(white_led.source_spct, target = 0.4, interpolate = TRUE)
find_wls(white_led.source_spct, target = c(0.3, 0.4))
find_wls(white_led.source_spct, target = c(0.3, 0.4), idfactor = "target")
find_wls(white_led.source_spct, target = c(0.3, 0.4), idfactor = TRUE)
find_wls(white_led.source_spct, target = "0.5max")
find_wls(white_led.source_spct, target = "0.05max")
find_wls(white_led.source_spct, target = "0.5range")

led.df <- as.data.frame(white_led.source_spct)
find_wls(led.df)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length")
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length",
         target = 0.4)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length",
         target = c(0.3, 0.4))
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length",
         target = 0.4, idfactor = "target")
```

`fit_peaks` *Refine position and value of extremes by fitting*

Description

Functions implementing fitting of peaks in a class-agnostic way. The fitting refines the location of peaks and value of peaks based on the location of maxima and minima supplied. This function is to be used together with `find_peaks()` or `find_valleys()`.

Usage

```
fit_peaks(
  x,
  peaks.idx,
  span,
  x.col.name = NULL,
  y.col.name,
  method,
  max.span = 5L,
  maximum = TRUE,
  keep.cols = NULL
)

fit_valleys(
  x,
  valleys.idx,
  span,
  x.col.name = NULL,
  y.col.name,
  method,
  max.span = 5L,
  maximum = FALSE,
  keep.cols = NULL
)
```

Arguments

`x`	generic_spct or data.frame object.
`peaks.idx`, `valleys.idx`	logical or integer Indexes into `x` selecting global or local extremes.
`span`	odd integer The span used when refining the location of maxima or minima of `x`.
`x.col.name`, `y.col.name`	character Name of the column of `x` on which to operate.
`method`	character The method to use for the fit.
`max.span`	odd integer The maximum number of data points used when when refining the location of maxima and minima.
`maximum`	logical A flag indicating whether to search for maxima or minima.
`keep.cols`	logical Keep unrecognized columns in data frames

Details

The only method currently implemented is `"spline"` based on a call to `splinefun` in a window of width `span` centred on each peak pointed at by `peaks.idx`. A spline fitted to a narrow window will usually locate the position of the peak in the column named by the argument passed to `x.col.name` better than estimating the true height of the peak in the column named by the argument passed to `y.col.name`.

Value

An R object of the same class as `x` containing the fitted values for the peaks, and optionally the unmodified values at the rows matching `peaks.idx` or `valleys.idx` for other retained columns.

Note

These functions are not meant for everyday use. Use option `refine.wl = TRUE` of methods `peaks()` and `valleys()` instead.

Examples

```
peaks <- find_peaks(sun.spct[["s.e.irrad"]], span = 31)
fit_peaks(sun.spct, peaks, span = 31,
          y.col.name = "s.e.irrad", method = "spline")
```

`fluence` *Fluence*

Description

Energy or photon fluence for one or more wavebands of a light source spectrum and a duration of exposure.

Usage

```
fluence(
  spct,
  w.band,
  unit.out,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
fluence(
  spct,
  w.band,
  unit.out,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
fluence(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
fluence(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	a list of `waveband` objects or a `waveband` object.
`unit.out`	character string with allowed values "energy", and "photon", or its alias "quantum".
`exposure.time`	lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, [W m-2 nm-1] -> [mol s-1 m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [mol d-1 m-2]

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_fluence()`, `e_irrad()`, `irrad()`, `q_fluence()`, `q_irrad()`

Examples

```
library(lubridate)
fluence(sun.spct,
        w.band = waveband(c(400,700)),
        exposure.time = lubridate::duration(3, "minutes") )
```

`formatted_range` *Compute range and format it*

Description

Compute the range of an R object, and format it as string suitable for printing.

Usage

```
formatted_range(x, na.rm = TRUE, digits = 3, nsmall = 2, collapse = "..")
```

Arguments

`x`	an R object
`na.rm`	logical, indicating if `NA`'s should be omitted.
`digits, nsmall`	numeric, passed to same name parameters of `format()`.
`collapse`	character, passed to same name parameter of `paste()`.

See Also

`range`, `format` and `paste`.

Examples

```
formatted_range(c(1, 3.5, -0.01))
```

`fscale` *Rescale a spectrum using a summary function*

Description

These methods return a spectral object of the same class as the one supplied as argument but with the spectral data rescaled based on a summary function `f` applied over a specific `range` of wavelengths and a `target` value for the summary value. When the object contains multiple spectra, the rescaling is applied separately to each spectrum.

Usage

```
fscale(x, ...)

## Default S3 method:
fscale(x, ...)

## S3 method for class 'source_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...
)

## S3 method for class 'response_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...
)

## S3 method for class 'filter_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  set.scaled = target == 1,
  ...
)

## S3 method for class 'reflector_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = NULL,
  set.scaled = target == 1,
  ...
)

## S3 method for class 'solute_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = NULL,
  set.scaled = target == 1,
  ...
)

## S3 method for class 'raw_spct'
fscale(x, range = NULL, f = "mean", target = 1, set.scaled = target == 1, ...)

## S3 method for class 'cps_spct'
fscale(x, range = NULL, f = "mean", target = 1, set.scaled = target == 1, ...)

## S3 method for class 'generic_spct'
```

```
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  col.names,
  ...
)

## S3 method for class 'source_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

```
## S3 method for class 'reflector_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = NULL,
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  col.names,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	An R object
`...`	additional named arguments passed down to `f`.
`range`	numeric. An R object on which `range()` returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm)
`f`	character string. "mean" or "total" for scaling so that this summary value becomes 1 for the returned object, or the name of a function taking `x` as first argument and returning a numeric value.
`target`	numeric A constant used as target value for scaling.
`unit.out`	character. Allowed values "energy", and "photon", or its alias "quantum".
`set.scaled`	logical or NULL Flag indicating if the data is to be marked as "scaled" or not.
`qty.out`	character. Allowed values "transmittance", and "absorbance".
`col.names`	character vector containing the names of columns or variables to which to apply the scaling.
`.parallel`	logical if TRUE, apply function in parallel, using parallel backend provided by foreach.
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

After scaling, calling the function passed as argument to `f` with the scaled spectrum as argument, will return the value passed as argument to `target`. **The default for `set.scaled` depends dynamically on the value passed to `target`.** Sometimes we rescale a spectrum to a "theoretical" value for the summary, while in other cases we rescale the spectrum to a real-world target value of, e.g., a reference energy irradiance. In the first case we say that the data are expressed in relative units, while in the second case we retain actual physical units. To indicate this, the default argument for 'set.scaled' is TRUE when `target == 1`, assuming the first of these two situations, and false otherwise, assuming the second situation. These defaults can be overriden with an explicit `logical` argument passed to `set.scaled`. Scaling overrides any previous normalization with the spectrum tagged as not normalized.

Method `fscale` is implemented for `solute_spct` objects but as the spectral data stored in them are a description of an intensive property of a substance, scaling is unlikely to useful. To represent solutions of specific concentrations of solutes, `filter_spct` objects should be used instead.

Value

A copy of the object passed as argument to `x` with the original spectral data values replaced with rescaled values, and the `"scaled"` attribute set to a list describing the scaling applied.

a new object of the same class as `x`.

Methods (by class)

- `fscale(default)`: Default for generic function
- `fscale(source_spct)`:
- `fscale(response_spct)`:
- `fscale(filter_spct)`:
- `fscale(reflector_spct)`:
- `fscale(solute_spct)`:
- `fscale(raw_spct)`:
- `fscale(cps_spct)`:
- `fscale(generic_spct)`:
- `fscale(source_mspct)`:
- `fscale(response_mspct)`:
- `fscale(filter_mspct)`:
- `fscale(reflector_mspct)`:
- `fscale(solute_mspct)`:
- `fscale(raw_mspct)`:
- `fscale(cps_mspct)`:
- `fscale(generic_mspct)`:

Important changes

Metadata describing the rescaling operation are stored in an attribute only if `set.scaled = TRUE` is passed to the call. The exact format and data stored in the attribute `"scaled"` has changed during the development history of the package. Spectra re-scaled with earlier versions will lack some information. To obtain the metadata in a consistent format irrespective of this variation use accessor `getScaling()`, which fills missing fields with `NA`.

See Also

Other rescaling functions: `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setNormalized()`, `setScaled()`

Examples

```
fscale(sun.spct)
fscale(sun.spct, f = "mean") # same as default
fscale(sun.spct, f = "mean", na.rm = TRUE)
fscale(sun.spct, range = c(400, 700)) # default is whole spectrum
fscale(sun.spct, f = "e_irrad", range = c(400, 700))
s400.spct <- fscale(sun.spct,
                    f = e_irrad,
                    range = c(400, 700),
                    target = 400) # a target in W m-2
s400.spct
e_irrad(s400.spct, c(400, 700))
```

fshift *Shift the scale of a spectrum using a summary function*

Description

The `fshift()` methods return a spectral object of the same class as the one supplied as argument but with the spectral data on a zero-shifted scale. A range of wavelengths is taken as a zero reference and the summary calculated with `f` for this waveband is substracted. This results in a zero shift (= additive correction) to the values in the returned object. Metadata attributes are retained unchanged.

Usage

```
fshift(x, ...)

## Default S3 method:
fshift(x, ...)

## S3 method for class 'source_spct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
fshift(x, range = c(wl_min(x), wl_min(x) + 10), f = "min", qty.out = NULL, ...)

## S3 method for class 'source_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'raw_spct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  qty.out = NULL,
  ...
)

## S3 method for class 'cps_spct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  qty.out = NULL,
  ...
)

## S3 method for class 'generic_spct'
fshift(x, range = c(wl_min(x), wl_min(x) + 10), f = "mean", col.names, ...)
```

```
## S3 method for class 'response_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  qty.out = NULL,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  col.names,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	An R object
`...`	additional named arguments passed down to `f`.
`range`	An R object on which `range()` returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm)
`f`	character string "mean", "min" or "max" for scaling so that this summary value becomes the origin of the spectral data scale in the returned object, or the name of a function taking `x` as first argument and returning a numeric value.
`unit.out`	character Allowed values "energy", and "photon", or its alias "quantum"
`qty.out`	character Allowed values "transmittance", and "absorbance"
`col.names`	character vector containing the names of columns or variables to which to apply the scale shift.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of `x` with the spectral data values replaced with values zero-shifted.

a new object of the same class as `x`.

Methods (by class)

- `fshift(default)`: Default for generic function
- `fshift(source_spct)`:
- `fshift(response_spct)`:
- `fshift(filter_spct)`:

- `fshift(reflector_spct)`:
- `fshift(source_mspct)`:
- `fshift(raw_spct)`:
- `fshift(cps_spct)`:
- `fshift(generic_spct)`:
- `fshift(response_mspct)`:
- `fshift(filter_mspct)`:
- `fshift(reflector_mspct)`:
- `fshift(raw_mspct)`:
- `fshift(cps_mspct)`:
- `fshift(generic_mspct)`:

Note

Method `fshift` is not implemented for `solute_spct` objects as the spectral data stored in them are a description of an intensive property of a substance. To represent solutions of specific concentrations of solutes, `filter_spct` objects can be used.

See Also

Other rescaling functions: `fscale()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setNormalized()`, `setScaled()`

`generic_mspct` *Collection-of-spectra constructor*

Description

Converts a list of spectral objects into a "multi spectrum" object by setting the class attribute of the list of spectra to the corresponding multi-spct class, check that components of the list belong to the expected class.

Usage

```
generic_mspct(
  l = NULL,
  class = "generic_spct",
  ncol = 1,
  byrow = FALSE,
  dim = c(length(l)%/%ncol, ncol)
)

calibration_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)
```

```
raw_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

cps_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

source_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

filter_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

reflector_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

object_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

solute_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

response_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

chroma_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)
```

Arguments

`l`	list of generic_spct or derived classes
`class`	character The multi spectrum object class or the expected class for the elements of l
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`dim`	integer vector of dimensions
`...`	ignored

Functions

- `calibration_mspct()`: Specialization for collections of `calibration_spct` objects.
- `raw_mspct()`: Specialization for collections of `raw_spct` objects.
- `cps_mspct()`: Specialization for collections of `cps_spct` objects.
- `source_mspct()`: Specialization for collections of `source_spct` objects.
- `filter_mspct()`: Specialization for collections of `filter_spct` objects.
- `reflector_mspct()`: Specialization for collections of `reflector_spct` objects.
- `object_mspct()`: Specialization for collections of `object_spct` objects.
- `solute_mspct()`: Specialization for collections of `solute_spct` objects.
- `response_mspct()`: Specialization for collections of `response_spct` objects.
- `chroma_mspct()`: Specialization for collections of `chroma_spct` objects.

Note

Setting class = source_spct or class = source_mspct makes no difference

Examples

```
filter_mspct(list(polyester.spct, yellow_gel.spct))
```

`getFilterProperties` *Get the "filter.properties" attribute*

Description

Function to read the "filter.properties" attribute of an existing filter_spct or a filter_mspct.

Usage

```
getFilterProperties(x, return.null, ...)

filter_properties(x, return.null, ...)

## Default S3 method:
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'summary_filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'generic_mspct'
getFilterProperties(x, return.null = FALSE, ..., idx = "spct.idx")
```

Arguments

`x`	a filter_spct object
`return.null`	logical If true, `NULL` is returned if the attribute is not set, otherwise the expected list is returned with all fields set to `NA`.
`...`	Allows use of additional arguments in methods for other classes.
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a list with fields named `"Rfr.constant"` $[/1]$, `"thickness"` $[m]$ and `"attenuation.mode"`. If the attribute is not set, and `return.null` is FALSE, a list with fields set to `NA` is returned, otherwise, `NULL`.

Methods (by class)

- `getFilterProperties(default)`: default
- `getFilterProperties(filter_spct)`: generic_spct
- `getFilterProperties(summary_filter_spct)`: summary_generic_spct
- `getFilterProperties(generic_mspct)`: filter_mspct

Note

The method for collections of spectra returns the a tibble with a column of lists.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
filter_properties(polyester.spct)
```

---

getHowMeasured *Get the "how.measured" attribute*

Description

Method to read the `"how.measured"` attribute of an R object.

Usage

```
getHowMeasured(x, ...)

how_measured(x, ...)

## Default S3 method:
getHowMeasured(x, ...)

## S3 method for class 'generic_spct'
getHowMeasured(x, ..., simplify = FALSE)

## S3 method for class 'summary_generic_spct'
getHowMeasured(x, ..., simplify = FALSE)
```

```
## S3 method for class 'data.frame'
getHowMeasured(x, ..., simplify = FALSE)

## S3 method for class 'generic_mspct'
getHowMeasured(x, ..., idx = "spct.idx", simplify = FALSE)
```

Arguments

`x`	an R object.
`...`	Allows use of additional arguments in methods for other classes.
`simplify`	logical If all members share the same attribute value return one copy instead of a data.frame.
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

character vector An object containing a verbal description of the data.

Methods (by class)

- `getHowMeasured(default)`: default
- `getHowMeasured(generic_spct)`: generic_spct
- `getHowMeasured(summary_generic_spct)`: summary_generic_spct
- `getHowMeasured(data.frame)`: data.frame
- `getHowMeasured(generic_mspct)`: generic_mspct

Note

The method for collections of spectra returns the a data frame with a column of character strings.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
how_measured(my.spct)
how_measured(my.spct) <- "Simulated with a radiation transfer model"
how_measured(my.spct)
how_measured(my.spct) <- NULL
how_measured(my.spct)
```

`getIdFactor` *Get the "idfactor" attribute*

Description

Function to read the `idfactor` attribute of an existing `generic_spct`.

Usage

```
getIdFactor(x)

id_factor(x)
```

Arguments

`x`	a generic_spct object

Value

character

Note

If x is not a `generic_spct` or an object of a derived class `NA` is returned.

See Also

Other idfactor attribute functions: `setIdFactor()`

Examples

```
id_factor(sun_evening.spct)
```

`getInstrDesc` *Get the "instr.desc" attribute*

Description

Function to query the `"instr.desc"` attribute of an existing `generic_spct` or derived-class object, or of a `summary_generic_spct` or derived-class object.

Usage

```
getInstrDesc(x)

instr_descriptor(x)
```

Arguments

x	a generic_spct object or a summary_generic_spct object.

Value

an object of class "instr_desc" derived from "list". The fields spectrometer.name, spectrometer.sn, bench.grating and bench.slit are always present, although may be set to NA. Additional fields can be present depending on the origin of the data.

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

Examples

```
valid.descriptor <- getInstrDesc(white_led.cps_spct)
class(valid.descriptor)
print(valid.descriptor)
print(str(valid.descriptor))

missing.descriptor <- getInstrDesc(white_body.spct)
class(missing.descriptor)
print(missing.descriptor)
print(str(missing.descriptor))
```

getInstrSettings *Get the "instr.settings" attribute*

Description

Function to extract the "instr.settings" attribute from generic_spct object or from a summary_generic_spct.

Usage

```
getInstrSettings(x)

instr_settings(x)
```

Arguments

x	a generic_spct object or a summary_generic_spct object.

Details

If x is derived from generic_spct or from summary_generic_spct, the value of attribute "instr.settings" is returned (NULL, if missing). Otherwise list() is returned.

Value

an object of class "instr_settings" derived from "list".

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

Examples

```
settings <- getInstrSettings(white_led.cps_spct)
class(settings)
print(settings)
print(str(settings))
```

---

getKType *Get the "K.type" attribute*

Description

Function to read the "K.type" attribute of an existing solute_spct object.

Usage

```
getKType(x)
```

Arguments

x	a solute_spct object

Value

character string

Note

If x is not a solute_spct or a summary_solute_spct object, NA is returned.

See Also

Other K attribute functions: `setKType()`

Examples

```
print("missing example")
```

---

getMspctVersion — *Get the "mspct.version" attribute*

Description

Function to read the "mspct.version" attribute of an existing generic_mspct object.

Usage

```
getMspctVersion(x)
```

Arguments

`x`	a generic_mspct object

Value

numeric value

Note

if x is not a `generic_mspct` object, `NA` is returned, and if it the attribute is missing, zero is returned with a warning.

---

getMultipleWl — *Get the "multiple.wl" attribute*

Description

Function to query the value of the `multiple.wl` attribute of an existing `generic_spct`.

Usage

```
getMultipleWl(x)

multiple_wl(x)
```

Arguments

`x`	a generic_spct object

Value

`integer` value, the value of attribute `multiple.wl`, or `NA` if the attribute is not set, or if `x` is not a `generic_spct` object or an object of a derived class.

See Also

Other multiple.wl attribute functions: `setMultipleWl()`

Examples

```
multiple_wl(sun.spct)
multiple_wl(sun_evening.spct)
```

`getNormalized` *Query the "normalized" and "normalization" attributes*

Description

Functions to read the "normalized" and "normalization" attributes of an existing generic_spct object.

Usage

```
getNormalized(x, .force.numeric = FALSE)

getNormalised(x, .force.numeric = FALSE)

getNormalization(x)

getNormalisation(x)
```

Arguments

`x`	a generic_spct object.
`.force.numeric`	logical If `TRUE` always silently return a numeric value, with `FALSE` encoded as zero, and character values as `NA`.

Details

Spectral data that has been normalized needs to be used diffferently in computations than data expresed in original units. These two functions make it possible to query if data stored in an object of class `generic_spct` or of a derived class contains data expressed in physical units or normalized. In the later case, it is possible to also query how the normalization was done.

Value

getNormalized() returns numeric or logical (possibly character for objects created with earlier versions); for collections of spectra, a named list, with one member for each spectrum. If x is not a generic_spct object, NA or a list with fields set to NAs is returned. Objects created with versions of package 'photobiology' earlier than 0.10.8 are lacking the detailed normalization metadata.

getNormalization() returns a list with five fields: norm.type, norm.wl, norm.factors, norm.cols, norm.range. For collections of spectra, a named list of lists, with one member list for each member of the collection of spectra. See setNormalized() for the values stored in the fields.

Note

getNormalised() is a synonym for this getNormalized() method.

See Also

Other rescaling functions: fscale(), fshift(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

Examples

```
getNormalized(sun.spct)
getNormalization(sun.spct)

sun_norm.spct <- normalize(sun.spct)

getNormalized(sun_norm.spct)
getNormalization(sun_norm.spct)

getNormalization(e2q(sun_norm.spct))

gel_norm.spct <- normalize(yellow_gel.spct)

getNormalized(gel_norm.spct)
getNormalization(gel_norm.spct)

# getNormalization(T2Afr(gel_norm.spct))
getNormalization(any2A(gel_norm.spct))
```

---

getScaled *Get the "scaled" attribute*

Description

Function to read the "scaled" attribute of an existing generic_spct object.

Usage

```
getScaled(x, .force.list = FALSE)

getScaling(x)
```

Arguments

x	a generic_spct object
.force.list	logical If TRUE always silently return a list, with FALSE encoded field multiplier = 1.

Value

logical

Note

if x is not a filter_spct object, NA is returned

See Also

Other rescaling functions: fscale(), fshift(), getNormalized(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

Examples

```
scaled.spct <- fscale(sun.spct)
getScaled(scaled.spct)
```

---

getSoluteProperties   *Get the "solute.properties" attribute*

Description

Function to read the "solute.properties" attribute of an existing solute_spct or a solute_mspct objects.

Usage

```
getSoluteProperties(x, return.null, ...)

solute_properties(x, return.null, ...)

## Default S3 method:
getSoluteProperties(x, return.null = FALSE, ...)
```

```
## S3 method for class 'solute_spct'
getSoluteProperties(x, return.null = FALSE, ...)

## S3 method for class 'summary_solute_spct'
getSoluteProperties(x, return.null = FALSE, ...)

## S3 method for class 'solute_mspct'
getSoluteProperties(x, return.null = FALSE, ..., idx = "spct.idx")
```

Arguments

`x`	solute_spct A spectrum of coefficients of attenuation.
`return.null`	logical If true, `NULL` is returned if the attribute is not set, otherwise the expected list is returned with all fields set to `NA`.
`...`	Allows use of additional arguments in methods for other classes.
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a `list` with fields named `"mass"`, `"formula"`, `"structure"`, `"name"` and `"ID"`. If the attribute is not set, and `return.null` is `FALSE`, a list with fields set to `NA` is returned, otherwise, `NULL`.

Methods (by class)

- `getSoluteProperties(default)`: default
- `getSoluteProperties(solute_spct)`: solute_spct
- `getSoluteProperties(summary_solute_spct)`: summary_solute_spct
- `getSoluteProperties(solute_mspct)`: solute_mspct

Note

The method for collections of spectra returns the a tibble with a column of lists.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
solute_properties(water.spct)
```

`getSpctVersion` *Get the "spct.version" attribute*

Description

Function to read the "spct.version" attribute of an existing generic_spct object.

Usage

```
getSpctVersion(x)
```

Arguments

`x`	a generic_spct object

Value

integer value

Note

if `x` is not a `generic_spct` object, `NA` is returned, and if it the attribute is missing, zero is returned with a warning.

`getTimeUnit` *Get the "time.unit" attribute of an existing source_spct object*

Description

Function to read the "time.unit" attribute

Usage

```
getTimeUnit(x, force.duration = FALSE)
```

Arguments

`x`	a source_spct object
`force.duration`	logical If TRUE a lubridate::duration is returned even if the object attribute is a character string, if no conversion is possible NA is returned.

Value

character string or a lubridate::duration

Note

if x is not a `source_spct` or a `response_spct` object, NA is returned

See Also

Other time attribute functions: `checkTimeUnit()`, `convertThickness()`, `convertTimeUnit()`, `setTimeUnit()`

Examples

```
getTimeUnit(sun.spct)
```

`getWhatMeasured` *Get the* `"what.measured"` *attribute*

Description

Method to read the `"what.measured"` attribute of an R object.

Usage

```
getWhatMeasured(x, ...)

what_measured(x, ...)

## Default S3 method:
getWhatMeasured(x, ...)

## S3 method for class 'generic_spct'
getWhatMeasured(x, ..., simplify = FALSE)

## S3 method for class 'summary_generic_spct'
getWhatMeasured(x, ..., simplify = FALSE)

## S3 method for class 'data.frame'
getWhatMeasured(x, ..., simplify = FALSE)

## S3 method for class 'generic_mspct'
getWhatMeasured(x, ..., idx = "spct.idx", simplify = FALSE)
```

Arguments

`x`	an R object.
`...`	Allows use of additional arguments in methods for other classes.
`simplify`	logical If all members share the same attribute value return one copy instead of a `data.frame`.

`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

`character` vector An object containing a description of the data. If `x` does not belong to a supported class `NA` is returned.

Methods (by class)

- `getWhatMeasured(default)`: default
- `getWhatMeasured(generic_spct)`: generic_spct
- `getWhatMeasured(summary_generic_spct)`: summary_generic_spct
- `getWhatMeasured(data.frame)`: data.frame
- `getWhatMeasured(generic_mspct)`: generic_mspct

Note

The method for collections of spectra returns the a `data.frame` with a column of character strings.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
what_measured(my.spct)
what_measured(my.spct) <- "Sun"
what_measured(my.spct)
what_measured(my.spct) <- NULL
what_measured(my.spct)
```

---

getWhenMeasured *Get the "when.measured" attribute*

Description

Method to read the `"when.measured"` attribute of an R object.

Usage

```
getWhenMeasured(x, ...)

when_measured(x, ...)

## Default S3 method:
getWhenMeasured(x, ...)

## S3 method for class 'generic_spct'
getWhenMeasured(x, as.df = FALSE, ..., simplify = FALSE)

## S3 method for class 'summary_generic_spct'
getWhenMeasured(x, as.df = FALSE, ..., simplify = FALSE)

## S3 method for class 'data.frame'
getWhenMeasured(x, as.df = FALSE, ..., simplify = FALSE)

## S3 method for class 'generic_mspct'
getWhenMeasured(x, ..., idx = "spct.idx", simplify = FALSE)
```

Arguments

`x`	an R object
`...`	Allows use of additional arguments in methods for other classes.
`as.df`	logical If `TRUE` return a data frame instead of a list, when the value stored in the attribute is a list.
`simplify`	logical If all members share the same attribute value return one copy instead of a data.frame.
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a `POSIXct` object with date and time, or named list of such objects, or, on user request, a data frame.

Methods (by class)

- `getWhenMeasured(default)`: default
- `getWhenMeasured(generic_spct)`: generic_spct
- `getWhenMeasured(summary_generic_spct)`: summary_generic_spct
- `getWhenMeasured(data.frame)`: data.frame
- `getWhenMeasured(generic_mspct)`: generic_mspct

Note

If `x` is not an object of one of the supported classes, `NA` is returned.

The method for collections of spectra returns a tibble with the times expressed in TZ = "UTC".

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

Examples

```
my.spct <- sun.spct
when_measured(my.spct)
when_measured(my.spct) <- lubridate::ymd_hms("2020-01-01 08:00:00")
when_measured(my.spct)
when_measured(my.spct) <- NULL
when_measured(my.spct)
```

---

getWhereMeasured *Get the "where.measured" attribute*

Description

Method to read the "where.measured" attribute of generic_spct, generic_mspct, summary_generic_spct, data.frame or a derived-class object.

Usage

```
getWhereMeasured(x, ...)

where_measured(x, ...)

## Default S3 method:
getWhereMeasured(x, ...)

## S3 method for class 'generic_spct'
getWhereMeasured(x, ..., simplify = FALSE)

## S3 method for class 'summary_generic_spct'
getWhereMeasured(x, ..., simplify = FALSE)

## S3 method for class 'generic_mspct'
getWhereMeasured(
  x,
  ...,
  idx = "spct.idx",
  .bind.geocodes = TRUE,
  simplify = FALSE
)

## S3 method for class 'data.frame'
getWhereMeasured(x, ...)
```

Arguments

`x`	a generic_spct object
`...`	Allows use of additional arguments in methods for other classes.
`simplify`	logical If all members share the same attribute value return one copy instead of a data.frame.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.bind.geocodes`	logical In the case of collections of spectra if `.bind.geocodes` = `TRUE`, the default, the returned value is a single geocode with one row for each member spectrum. Otherwise the individual geocode data frames are returned in a list column within a tibble.

Value

a data.frame with a single row and at least columns "lon" and "lat", unless expand is set to `FALSE`.

Methods (by class)

- `getWhereMeasured(default)`: default
- `getWhereMeasured(generic_spct)`: generic_spct
- `getWhereMeasured(summary_generic_spct)`: summary_generic_spct
- `getWhereMeasured(generic_mspct)`: generic_mspct
- `getWhereMeasured(data.frame)`: data.frame

Note

If x is not a `generic_spct` or an object of a derived class `NA` is returned.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
where_measured(my.spct)
where_measured(my.spct) <- data.frame(lon = 0, lat = -60)
where_measured(my.spct)
where_measured(my.spct) <- NULL
where_measured(my.spct)
```

---

get_attributes *Get the metadata attributes*

Description

Method returning attributes of an object of class generic_spct or derived, or of class waveband. Only attributes defined and/or set by package 'photobiology' for objects of the corresponding class are returned. Parameter `which` can be used to subset the list of attributes.

Usage

```
get_attributes(x, which, ...)

## S3 method for class 'generic_spct'
get_attributes(x, which = NULL, allowed = all.attributes, ...)

## S3 method for class 'source_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'filter_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'reflector_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'object_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'solute_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'waveband'
get_attributes(x, which = NULL, ...)
```

Arguments

`x`	a generic_spct object.
`which`	character vector Names of attributes to retrieve.

...	currently ignored
allowed	character vector Names of attributes accepted by which.

Details

Vectors of character strings passed as argument to which are parsed so that if the first member string is "-" the remaining members are removed from the allowed; and if it is "=" the remaining members are used if in allowed. If the first member is none of these three strings, the behaviour is the same as if the first string is "=". If which is NULL all the attributes in allowed are used. The string "" means no attributes, and has precedence over any other values in the character vector. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

Named list of attribute values.

Methods (by class)

- get_attributes(generic_spct): generic_spct
- get_attributes(source_spct): source_spct
- get_attributes(filter_spct): filter_spct
- get_attributes(reflector_spct): reflector_spct
- get_attributes(object_spct): object_spct
- get_attributes(solute_spct): solute_spct
- get_attributes(waveband): waveband

See Also

select_spct_attributes

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

---

get_peaks *Get peaks and valleys from a spectrum*

Description

These functions "get" (or extract) peaks (maxima) and valleys (minima) in two vectors, usually a spectral quantity and wavelength, using a user selectable span for window width and global and local (within moving window) size thresholds. They also generate character values for x.

Usage

```
get_peaks(
  x,
  y,
  global.threshold = 0,
  span = 5,
  strict = TRUE,
  x_unit = "",
  x_digits = 3,
  na.rm = FALSE
)

get_valleys(
  x,
  y,
  global.threshold = 0,
  span = 5,
  strict = TRUE,
  x_unit = "",
  x_digits = 3,
  na.rm = FALSE
)
```

Arguments

x, y	numeric
global.threshold	numeric A value belonging to class "AsIs" is interpreted as an absolute minimum height or depth expressed in data units. A bare numeric value (normally between 0.0 and 1.0), is interpreted as relative to threshold.range. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are ignored. A bare negative numeric value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If global.threshold = NULL, no threshold is applied and all peaks returned.
span	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width span centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. span = NULL extends the span to the whole length of x.
strict	logical flag: if TRUE, an element must be strictly greater than all other values in its window to be considered a peak.
x_unit	character Vector of texts to be pasted at end of labels built from x value at peaks.
x_digits	numeric Number of significant digits in wavelength label.
na.rm	logical indicating whether NA values should be stripped before searching for peaks.

Details

As find_valleys, peaks and valleys call find_peaks to search for peaks and valleys, this explanation applies to the four functions. It also applies to stat_peaks and stat_valleys. Function find_peaks is a wrapper built onto function peaks from **splus2R**, adds support for peak height thresholds and handles span = NULL and non-finite (including NA) values differently than splus2R::peaks. Instead of giving an error when na.rm = FALSE and x contains NA values, NA values are replaced with the smallest finite value in x. span = NULL is treated as a special case and selects max(x). Passing strict = TRUE ensures that non-unique global and within window maxima are ignored, and can result in no peaks being returned.

Two tests make it possible to ignore irrelevant peaks. One test (global.threshold) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (local.threshold) is available when the window defined by 'span' does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by the argument passed to local.reference. Parameter threshold.range determines how the values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being returned.

The local.threshold argument is used *as is* when local.reference is "median" or "farthest", i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a local.threshold that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with local.reference arguments "median.log", "farthest.log", "median.sqrt", and "farthest.sqrt" as arguments for local.reference.

While functions find_peaks and find_valleys accept as input a numeric vector and return a logical vector, methods peaks and valleys accept as input different R objects, including spectra and collections of spectra and return a subset of the object. These methods are implemented using calls to functions find_peaks, find_valleys and fit_peaks.

Value

A data frame with variables w.length and s.irrad with their values at the peaks or valleys plus a character variable of labels.

Note

The use of these two functions is deprecated. They are retained for backwards compatibility and will be removed in the near future.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

`green_leaf.spct` *Green birch leaf reflectance.*

Description

A dataset of spectral reflectance expressed as a fraction of one.

Usage

```
green_leaf.spct
```

Format

A `reflector_spct` object with 226 rows and 2 variables

Details

- w.length (nm)
- Rfr (0..1)

References

Aphalo, P. J. & Lehto, T. Effects of light quality on growth and N accumulation in birch seedlings Tree Physiology, 1997, 17, 125-132

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
green_leaf.spct
```

head_tail — *Return the First and Last Parts of an Object*

Description

Returns the first and last "parts" (rows or members) of a spectrum, dataframe, vector, function, table or ftable. In other words, the combined output from methods `head` and `tail`.

Usage

```
head_tail(x, n, ...)

## Default S3 method:
head_tail(x, n = 3L, ...)

## S3 method for class 'data.frame'
head_tail(x, n = 3L, ...)

## S3 method for class 'matrix'
head_tail(x, n = 3L, ...)

## S3 method for class '`function`'
head_tail(x, n = 6L, ...)

## S3 method for class 'table'
head_tail(x, n = 6L, ...)

## S3 method for class 'ftable'
head_tail(x, n = 6L, ...)
```

Arguments

`x`	an R object.
`n`	integer. If positive, `n` rows or members in the returned object are copied from each of "head" and "tail" of `x`. If negative, all except `n` elements of `x` from each of "head" and "tail" are returned.
`...`	arguments to be passed to or from other methods.

Details

The value returned by `head_tail()` is equivalent to row binding the the values returned by `head()` and `tail()`, although not implemented in this way. The same specializations as defined in package 'utils' for `head()` and `tail()` have been implemented.

Value

An object (usually) like `x` but smaller, except when `n = 0`. For `ftable` objects `x`, a transformed `format(x)`.

Methods (by class)

- `head_tail(default)`:
- `head_tail(data.frame)`:
- `head_tail(matrix)`:
- ``head_tail(`function`)``:
- `head_tail(table)`:
- `head_tail(ftable)`:

Note

For some types of input, like functions, the output may be confusing, however, we have opted for consistency with existing functions. The code is in part a revision of that of `head()` and `tail()` from package 'utils'. This method is especially useful when checking spectral data, as both ends are of interest.

`head_tail()` methods for function, table and ftable classes, are wrappers for head() method.

See Also

`head`, and compare the examples and the values returned to the examples below.

Examples

```
head_tail(1:20)
head_tail(1:20, 12)
head_tail(1:20, -7)
head_tail(1:20, -10)
head_tail(letters)
head_tail(sun.spct)
head_tail(sun.spct, 6)
head_tail(sun.data)
head_tail(as.matrix(sun.data))
head_tail(sun_evening.spct)
head_tail(sun_evening.mspct, 1L)
```

`illuminance` *Irradiance*

Description

Computes illuminance (lux), or the luminous flux incident on a surface, from spectral irradiance stored in a `source_spct` object.

Usage

```
illuminance(spct, std, scale.factor, allow.scaled, ...)

## Default S3 method:
illuminance(spct, std, scale.factor, allow.scaled, ...)

## S3 method for class 'source_spct'
illuminance(
  spct,
  std = "CIE2deg",
  scale.factor = 1,
  allow.scaled = FALSE,
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
illuminance(
  spct,
  std = "CIE2deg",
  scale.factor = 1,
  allow.scaled = FALSE,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`std`	character The luminous efficiency function to use, `"CIE2deg"` or `"CIE10deg"`.
`scale.factor`	numeric vector of length 1, or the character string `exposure`.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
`...`	other arguments (possibly ignored)
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach.

`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector of length one in the case of methods for individual spectra. A `data.frame` in the case of collections of spectra, containing one column with illuminance, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

The `time.unit` attribute is always second. Units are as follows: if time.unit of the argument passed to `spct` is second, [W m-2 nm-1] -> [lx], otherwise average value [lx] for the period unless `exposure = TRUE`.

Methods (by class)

- `illuminance(default)`: Default for generic function
- `illuminance(source_spct)`: Calculates illuminance from a `source_spct` object.
- `illuminance(source_mspct)`: Calculates illuminance from a `source_mspct` object.

Note

Formal parameter `allow.scaled` is used internally for calculation of ratios, as rescaling and normalization do not invalidate the calculation of ratios within one spectrum.

References

Stockman, A. (2019) Cone fundamentals and CIE standards. *Current Opinion in Behavioral Sciences*, 30, 87-93. doi:10.1016/j.cobeha.2019.06.005

Examples

```
illuminance(sun.spct)
illuminance(sun.daily.spct)
illuminance(sun.daily.spct, scale.factor = "exposure")
illuminance(sun.daily.spct, scale.factor = 1e-3)
```

`insert_hinges` *Insert wavelength values into spectral data.*

Description

Inserting wavelengths values immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data have a large wavelength step size.

Usage

```
insert_hinges(x, y, h)
```

Arguments

x	numeric vector (sorted in increasing order)
y	numeric vector
h	a numeric vector giving the wavelengths at which the y values should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0))

Value

a data.frame with variables x and y. Unless the hinge values were already present in y, each inserted hinge, expands the vectors returned in the data frame by one value.

Note

Insertion is a costly operation but I have tried to optimize this function as much as possible by avoiding loops. Earlier this function was implemented in C++, but a bug was discovered and I have now rewritten it using R.

See Also

Other low-level functions operating on numeric vectors.: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

```
with(sun.data,
    insert_hinges(w.length, s.e.irrad,
       c(399.99, 400.00, 699.99, 700.00)))
```

---

insert_spct_hinges *Insert new wavelength values into a spectrum*

Description

Insert new wavelength values into a spectrum interpolating the corresponding spectral data values.

Usage

```
insert_spct_hinges(spct, hinges = NULL, byref = FALSE)
```

Arguments

`spct`	an object of class "generic_spct"
`hinges`	numeric vector of wavelengths (nm) at which the s.irrad should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0))
`byref`	logical indicating if new object will be created by reference or by copy of spct

Value

a generic_spct or a derived type with variables `w.length` and other numeric variables.

Note

Inserting wavelengths values "hinges" immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data has a large wavelength step size.

Examples

```
insert_spct_hinges(sun.spct, c(399.99,400.00,699.99,700.00))
insert_spct_hinges(sun.spct,
                   c(199.99,200.00,399.50,399.99,400.00,699.99,
                         700.00,799.99,1000.00))
```

`integrate_spct` *Integrate spectral data.*

Description

This function gives the result of integrating spectral data over wavelengths.

Usage

```
integrate_spct(spct)
```

Arguments

`spct`	generic_spct

Value

One or more numeric values with no change in scale factor: e.g. [W m-2 nm-1] -> [W m-2]. Each value in the returned vector corresponds to a variable in the spectral object, except for wavelength. For non-numeric variables the returned value is `NA`.

Examples

```
integrate_spct(sun.spct)
```

integrate_xy *Gives irradiance from spectral irradiance.*

Description

This function gives the result of integrating spectral irradiance over wavelengths.

Usage

```
integrate_xy(x, y)
```

Arguments

x	numeric vector.
y	numeric vector.

Value

a single numeric value with no change in scale factor: e.g. [W m-2 nm-1] -> [W m-2]

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data, integrate_xy(w.length, s.e.irrad))
```

interpolate_spct *Map a spectrum to new wavelength values.*

Description

This function gives the result of interpolating spectral data from the original set of wavelengths to a new one.

Usage

```
interpolate_spct(spct, w.length.out = NULL, fill = NA, length.out = NULL)

interpolate_mspct(
  mspct,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	generic_spct
`w.length.out`	numeric vector of wavelengths (nm)
`fill`	a value to be assigned to out of range wavelengths
`length.out`	numeric value
`mspct`	an object of class "generic_mspct"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Depending on the extent of the data natural spline interpolation or linear interpolation are used. In the first case a call to `spline` with `method = "natural"` is used when 25 or fewer distinct wavelengths are available as input, or if the wavelengths in the output are more than three times those in the input. In the second case, a call to `approx` is used.

If `w.length.out` is a numeric vector and `length.out = NULL`, it directly gives the target wavelengths for interpolation. If it is `NULL`, and `length.out` is an integer value evenly spaced wavelength values covering the same wavelength range as in the input are generated. If `w.length.out` is a numeric vector and `length.out` is an integer value, `length.out` evenly spaced wavelengths covering the wavelength range of `w.length.out` are generated. *Extrapolation is not supported.*

With default `fill = NA` if the output exceeds the wavelength range of the input, extrapolated values are filled with `NA` values. With `fill = NULL` wavelengths outside the wavelength range of input data are discarded. A numerical value can be also be provided as fill. While `interpolate_spectrum` supports interpolation of a single numeric vector, `interpolate_wl` applies, one at a time, interpolation to all numeric columns found in `x`.

Value

A new spectral object of the same class as argument `spct` with a different number of rows than `x`, different `w.length` values and new numeric values for spectral data obtained by interpolation.

Examples

```
interpolate_spct(sun.spct, 400:500, NA)
interpolate_spct(sun.spct, 400:500, NULL)
interpolate_spct(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_spct(sun.spct, c(400,500), length.out=201)
```

---

interpolate_spectrum *Calculate spectral values at a different set of wavelengths*

Description

Interpolate/re-express spectral irradiance (or other spectral quantity) values at new wavelengths values. This is a low-level function operating on numeric vectors and called by higher level functions in the package, such as mathematical operators for classes for spectral data.

Usage

```
interpolate_spectrum(w.length.in, s.irrad, w.length.out, fill = NA, ...)
```

Arguments

`w.length.in`	numeric vector of wavelengths (nm).
`s.irrad`	a numeric vector of spectral values.
`w.length.out`	numeric vector of wavelengths (nm).
`fill`	a value to be assigned to out of range wavelengths.
`...`	additional arguments passed to `spline()`.

Details

Depending on the extent of the data natural spline interpolation or linear interpolation are used. In the first case a call to `spline` with `method = "natural"` is used when 25 or fewer distinct wavelengths are available as input, or if the wavelengths in the output are more than three times those in the input. In the second case, a call to `approx` is used.

If `w.length.out` is a numeric vector and `length.out = NULL`, it directly gives the target wavelengths for interpolation. If it is `NULL`, and `length.out` is an integer value evenly spaced wavelength values covering the same wavelength range as in the input are generated. If `w.length.out` is a numeric vector and `length.out` is an integer value, `length.out` evenly spaced wavelengths covering the wavelength range of `w.length.out` are generated. *Extrapolation is not supported.*

With default `fill = NA` if the output exceeds the wavelength range of the input, extrapolated values are filled with `NA` values. With `fill = NULL` wavelengths outside the wavelength range of input data are discarded. A numerical value can be also be provided as fill. While `interpolate_spectrum` supports interpolation of a single numeric vector, `interpolate_wl` applies, one at a time, interpolation to all numeric columns found in `x`.

Value

a numeric vector of interpolated spectral values.

See Also

spline and approx.

Other low-level functions operating on numeric vectors.: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

```
my.w.length <- 300:700
with(sun.data, interpolate_spectrum(w.length, s.e.irrad, my.w.length))
```

---

interpolate_wl *Map spectra to new wavelength values.*

Description

This function returns the result of interpolating spectral data from the original set of wavelengths to a new one.

Usage

```
interpolate_wl(x, w.length.out, fill, length.out, ...)

## Default S3 method:
interpolate_wl(x, w.length.out, fill, length.out, ...)

## S3 method for class 'generic_spct'
interpolate_wl(x, w.length.out = NULL, fill = NA, length.out = NULL, ...)

## S3 method for class 'generic_mspct'
interpolate_wl(
  x,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object
`w.length.out`	numeric vector of wavelengths (nm)
`fill`	a value to be assigned to out of range wavelengths
`length.out`	numeric value
`...`	not used
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Depending on the extent of the data natural spline interpolation or linear interpolation are used. In the first case a call to `spline` with `method = "natural"` is used when 25 or fewer distinct wavelengths are available as input, or if the wavelengths in the output are more than three times those in the input. In the second case, a call to `approx` is used.

If `w.length.out` is a numeric vector and `length.out = NULL`, it directly gives the target wavelengths for interpolation. If it is `NULL`, and `length.out` is an integer value evenly spaced wavelength values covering the same wavelength range as in the input are generated. If `w.length.out` is a numeric vector and `length.out` is an integer value, `length.out` evenly spaced wavelengths covering the wavelength range of `w.length.out` are generated. *Extrapolation is not supported.*

With default `fill = NA` if the output exceeds the wavelength range of the input, extrapolated values are filled with `NA` values. With `fill = NULL` wavelengths outside the wavelength range of input data are discarded. A numerical value can be also be provided as fill. While `interpolate_spectrum` supports interpolation of a single numeric vector, `interpolate_wl` applies, one at a time, interpolation to all numeric columns found in `x`.

Value

A new spectral object or collection of spectral objects, of the same class as argument `x`. Each spectrum returned with more or fewer rows than in `x`, the requested new `w.length` values and new numeric values for spectral quantities, obtained by interpolation.

Methods (by class)

- `interpolate_wl(default)`: Default for generic function
- `interpolate_wl(generic_spct)`: Interpolate wavelength in an object of class "generic_spct" or derived.
- `interpolate_wl(generic_mspct)`: Interpolate wavelength in an object of class "generic_mspct" or derived.

Examples

```
interpolate_wl(sun.spct, 400:500, NA)
interpolate_wl(sun.spct, 400:500, NULL)
interpolate_wl(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_wl(sun.spct, c(400,500), length.out=201)
```

`irrad` *Irradiance*

Description

This function returns the irradiance for a given waveband of a light source spectrum.

Usage

```
irrad(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
irrad(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
irrad(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  return.tb = FALSE,
  ...
)

## S3 method for class 'source_mspct'
irrad(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are summarized.
`unit.out`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`quantity`	character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
`time.unit`	character or lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.

`use.cached.mult`	logical indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands. If NULL, default is chosen based on data.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to `spct` trigger an error.
`...`	other arguments (possibly ignored)
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`return.tb`	logical Flag forcing a tibble to be always returned, even for a single spectrum as argumnet to `spct`. The default is `FALSE` for backwards compatibility.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach.
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of a `_spct` object containing a single spectrum and `return.tb = FALSE`. The vector has one member one value for each `waveband` passed to parameter `w.band`. In all other cases a `tibble`, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

If `naming = "long"` the names generated reflect both quantity and waveband, if `naming = "short"`, names are based only on the wavebands, and if `naming = "none"` the returned vector has no names.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The `time.unit` attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, [W m-2 nm-1] -> [mol s-1 m-2] or [W m-2 nm-1] -> [W m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [mol d-1 m-2] or [J d-1 m-2 nm-1] -> [J m-2]

Note

Formal parameter `allow.scaled` is used internally for calculation of ratios, as rescaling and normalization do not invalidate the calculation of ratios.

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_fluence()`, `e_irrad()`, `fluence()`, `q_fluence()`, `q_irrad()`

Examples

```
irrad(sun.spct, waveband(c(400,700)))
irrad(sun.spct, waveband(c(400,700)), "energy")
irrad(sun.spct, waveband(c(400,700)), "photon")
irrad(sun.spct, split_bands(c(400,700), length.out = 3))
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")
```

`irradiance` *Photon or energy irradiance from spectral energy or photon irradiance.*

Description

Energy or photon irradiance for one or more wavebands of a radiation spectrum.

Usage

```
irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.out = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric Vector of wavelength [nm].
`s.irrad`	numeric vector of spectral (energy) irradiances [$W\ m^{-2}\ nm^{-1}$].
`w.band`	waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are summarized.

`unit.out, unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is `TRUE`.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value or a vector of numeric values with no change in scale factor: $[mol\, s^{-1}\, sm^{-2}\, nm^{-1}]$ yields $[mol\, s^{-1}\, sm^{-2}]$

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum()` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector. The is no reason for setting `use.cpp.code=FALSE` other than for testing the improvement in speed, or in cases where there is no suitable C++ compiler for building the package.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data, irradiance(w.length, s.e.irrad, new_waveband(400,700), "photon"))
```

`is.generic_mspct` *Query class of spectrum objects*

Description

Functions to check if an object is of a given type of spectrum, or coerce it if possible.

Usage

```
is.generic_mspct(x)

is.calibration_mspct(x)

is.raw_mspct(x)

is.cps_mspct(x)

is.source_mspct(x)

is.response_mspct(x)

is.filter_mspct(x)

is.reflector_mspct(x)

is.object_mspct(x)

is.solute_mspct(x)

is.chroma_mspct(x)

is.any_mspct(x)
```

Arguments

`x`	an R object.

Value

These functions return `TRUE` if its argument is a of the queried type of spectrum and `FALSE` otherwise.

Note

Derived types also return TRUE for a query for a base type such as `generic_mspct`.

Examples

```
my.mspct <- filter_mspct(list(polyester.spct, yellow_gel.spct))
is.any_mspct(my.mspct)
is.filter_mspct(my.mspct)
is.source_mspct(my.mspct)
```

is.generic_spct *Query class of spectrum objects*

Description

Functions to query whether an object is of a given type of spectrum.

Usage

```
is.generic_spct(x)

is.raw_spct(x)

is.calibration_spct(x)

is.cps_spct(x)

is.source_spct(x)

is.response_spct(x)

is.filter_spct(x)

is.reflector_spct(x)

is.object_spct(x)

is.solute_spct(x)

is.chroma_spct(x)

is.any_spct(x)
```

Arguments

x	an R object.

Value

A logical value, TRUE if the argument passed to x is an object of the queried type of spectrum and FALSE otherwise.

Note

Derived types also return TRUE for a query for a base type such as generic_spct, following R's practice.

Examples

```
is.source_spct(sun.spct)
is.filter_spct(sun.spct)
is.generic_spct(sun.spct)
is.generic_spct(sun.spct)

is.source_spct(sun.spct)
is.filter_spct(sun.spct)
is.generic_spct(sun.spct)
is.generic_spct(sun.spct)
```

`is.old_spct` *Query if an object has old class names*

Description

Query if an object has old class names Query if an object has old class names as used in photobiology (>= 0.6.0).

Usage

```
is.old_spct(object)
```

Arguments

`object`	an R object

Value

logical

See Also

Other upgrade from earlier versions: `upgrade_spct()`, `upgrade_spectra()`

`is.summary_generic_spct` *Query class of spectrum summary objects*

Description

Functions to check if an object is of a given type of spectrum, or coerce it if possible.

Usage

```
is.summary_generic_spct(x)

is.summary_raw_spct(x)

is.summary_cps_spct(x)

is.summary_source_spct(x)

is.summary_response_spct(x)

is.summary_filter_spct(x)

is.summary_reflector_spct(x)

is.summary_object_spct(x)

is.summary_solute_spct(x)

is.summary_chroma_spct(x)

is.any_summary_spct(x)
```

Arguments

`x`	an R object.

Value

These functions return `TRUE` if its argument is a of the queried type of spectrum and `FALSE` otherwise.

Note

Derived types also return TRUE for a query for a base type such as `generic_spct`.

Examples

```
sm <- summary(sun.spct)
is.summary_source_spct(sm)
```

`is.waveband` *Query if it is a waveband*

Description

Functions to check if an object is waveband.

Usage

```
is.waveband(x)
```

Arguments

`x`	any R object

Value

is.waveband returns TRUE if its argument is a waveband and FALSE otherwise.

`isValidInstrDesc` *Check the "instr.desc" attribute*

Description

Function to validate the `"instr.settings"` attribute of an existing `generic_spct` object or `summary_generic_spct` object.

Usage

```
isValidInstrDesc(x)
```

Arguments

`x`	a `generic_spct` object or a `summary_generic_spct` object.

Details

Test if at least one of instrument name (field `spectrometer.name`) or serial number (field `spectrometer.sn`) is found in the value of the R attribute `"instr.desc"` of `x`. `FALSE` is silently returned if `x` does not belong to a class derived from class `generic_spct` or from class `summary_generic_spct`, or if it is derived from these classes but the attribute is not set.

Value

A `logical` vector of length one.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrSettings()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
isValidInstrDesc(white_led.cps_spct)
isValidInstrDesc(white_body.spct)
```

isValidInstrSettings *Check the "instr.settings" attribute*

Description

Function to validate the "`instr.settings`" attribute of an existing `generic_spct` or `summary_generic_spct` object.

Usage

```
isValidInstrSettings(x)
```

Arguments

`x`	a `generic_spct` object or a `summary_generic_spct` object.

Value

logical TRUE if at least the integration time is found in the metadata attribute. If `x` is not a `generic_spct` or a `summary_generic_spct` object, `NA` is returned.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `select_spct_attributes()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

is_absorbance_based *Query if a spectrum contains absorbance or transmittance data*

Description

Functions to query if an filter spectrum contains spectral absorbance data or spectral transmittance data.

Usage

```
is_absorbance_based(x)

is_absorptance_based(x)

is_transmittance_based(x)
```

Arguments

`x`	an R object

Value

`is_absorbance_based` returns a logical value, `TRUE` if its argument is a `filter_spct` object that contains spectral absorbance data and `FALSE` otherwise, but returns `NA` for any other R object, including those belonging other `generic_spct`-derived classes.

`is_absorptance_based` returns a logical value, if its argument is a `filter_spct` object, `TRUE` if it contains data as spectral absorptance and `FALSE` otherwise, but returns `NA` for any other R object, including those belonging other `generic_spct`-derived classes.

`is_transmittance_based` returns TRUE if its argument is a `filter_spct` object that contains spectral transmittance data and FALSE if it does not contain such data, but returns NA for any other R object, including those belonging other `generic_spct`-derived classes.

See Also

Other query units functions: `is_mole_based()`, `is_photon_based()`

Examples

```
is_absorbance_based(polyester.spct)
my.spct <- T2A(polyester.spct)
is.filter_spct(my.spct)
is_absorbance_based(my.spct)

is_absorptance_based(polyester.spct)

is_transmittance_based(polyester.spct)
```

`is_effective` *Is an R object "effective"*

Description

A generic function for querying if a biological spectral weighting function (BSWF) has been applied to an object or is included in its definition.

Usage

```
is_effective(x)

## Default S3 method:
is_effective(x)

## S3 method for class 'waveband'
is_effective(x)

## S3 method for class 'generic_spct'
is_effective(x)

## S3 method for class 'source_spct'
is_effective(x)

## S3 method for class 'summary_generic_spct'
is_effective(x)

## S3 method for class 'summary_source_spct'
is_effective(x)
```

Arguments

`x`	an R object

Value

A `logical`.

Methods (by class)

- `is_effective(default)`: Default method.
- `is_effective(waveband)`: Is a `waveband` object defining a method for calculating effective irradiance.
- `is_effective(generic_spct)`: Does a `source_spct` object contain effective spectral irradiance values.
- `is_effective(source_spct)`: Does a `source_spct` object contain effective spectral irradiance values.
- `is_effective(summary_generic_spct)`: Method for "summary_generic_spct".
- `is_effective(summary_source_spct)`: Method for "summary_source_spct".

See Also

Other waveband attributes: `labels()`, `normalization()`

Examples

```
is_effective(summary(sun.spct))
```

`is_mole_based` *Query if a spectrum contains mole or mass based data*

Description

Functions to check if an solute attenuation spectrum contains coefficients on expressed on mole of mass base.

Usage

```
is_mole_based(x)

is_mass_based(x)
```

Arguments

`x`	an R object

Value

`is_mole_based` returns TRUE if its argument is a `solute_spct` object that contains spectral `K.mole` data and `FALSE` if it contains `K.mass` data, but returns NA for any other R object, including those belonging other `generic_spct`-derived classes. `is_mass_based` returns the complement of `is_mole_based`.

See Also

Other query units functions: `is_absorbance_based()`, `is_photon_based()`

Examples

```
print("missing example")
```

`is_normalized` *Query whether a generic spectrum has been normalized.*

Description

This function tests a `generic_spct` object for an attribute that signals whether the spectral data has been normalized or not after the object was created.

Usage

```
is_normalized(x)

is_normalised(x)
```

Arguments

`x`	An R object.

Value

A `logical` value indicating if `x` is normalized or not, for collections of spectra, a named list with `logicals` as members. If `x` is not a `generic_spct` or `generic_mspct` object the value returned is `NA`.

Note

`is_normalised()` is a synonym for this `is_normalized()` method.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_scaled()`, `normalize()`, `setNormalized()`, `setScaled()`

`is_photon_based` *Query if a spectrum contains photon- or energy-based data.*

Description

Functions to query if `source_spct` and `response_spct` objects contain photon-based or energy-based data.

Usage

```
is_photon_based(x)

is_energy_based(x)
```

Arguments

x	any R object

Value

`is_photon_based` returns a logical value, `TRUE` if its argument is a `source_spct` or a `response_spct` object that contains photon base data and `FALSE` otherwise, but returns `NA` for any other R object, including those belonging other `generic_spct`-derived classes.

`is_energy_based` returns a logical value, `TRUE` if its argument is a `source_spct` or a `response_spct` object that contains energy base data and `FALSE` otherwise, but returns `NA` for any other R object, including those belonging other `generic_spct`-derived classes

See Also

Other query units functions: `is_absorbance_based()`, `is_mole_based()`

Examples

```
colnames(sun.spct)
is_photon_based(sun.spct)
my.spct <- sun.spct[ , c("w.length", "s.e.irrad")]
is.source_spct(my.spct)
is_photon_based(my.spct)

colnames(sun.spct)
is_energy_based(sun.spct)
my.spct <- sun.spct[ , c("w.length", "s.q.irrad")]
is.source_spct(my.spct)
is_energy_based(my.spct)
```

`is_scaled` *Query whether a generic spectrum has been scaled*

Description

This function tests a `generic_spct` object for an attribute that signals whether the spectral data has been rescaled or not after the object was created.

Usage

```
is_scaled(x)
```

Arguments

x	An R object.

Value

A `logical` value. If `x` is not scaled or `x` is not a `generic_spct` object the value returned is `FALSE`.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `normalize()`, `setNormalized()`, `setScaled()`

Examples

```
scaled.spct <- fscale(sun.spct)
is_scaled(sun.spct)
is_scaled(scaled.spct)
```

`is_tagged` *Query if a spectrum is tagged*

Description

Functions to check if an spct object contains tags.

Usage

```
is_tagged(x)
```

Arguments

`x`	any R object

Value

`is_tagged` returns a logical value, `TRUE` if its argument is a a spectrum that contains tags and `FALSE` if it is an untagged spectrum, but returns `NA` for any other R object.

See Also

Other tagging and related functions: `tag()`, `untag()`, `wb2rect_spct()`, `wb2spct()`, `wb2tagged_spct()`

Examples

```
is_tagged(sun.spct)
```

join_mspct *Join all spectra in a collection*

Description

Join all the spectra contained in a homogeneous collection, returning a data frame with spectral-data columns named according to the names of the spectra in the collection. By default a full join is done within the overlapping range of wavelengths, after interpolating the spectra to a shared set of wavelength values, and discarding data for wavelength not shared. Alternatively, filling the spectral data for wavelengths outside the overlapping range with with `NA` when data is not available.

Usage

```
join_mspct(x, type, ...)

## Default S3 method:
join_mspct(x, type = "full", ...)

## S3 method for class 'generic_mspct'
join_mspct(x, type = "full", col.name, validate.names = TRUE, ...)

## S3 method for class 'source_mspct'
join_mspct(x, type = "full", unit.out = "energy", validate.names = TRUE, ...)

## S3 method for class 'response_mspct'
join_mspct(x, type = "full", unit.out = "energy", validate.names = TRUE, ...)

## S3 method for class 'filter_mspct'
join_mspct(
  x,
  type = "full",
  qty.out = "transmittance",
  validate.names = TRUE,
  ...
)

## S3 method for class 'reflector_mspct'
join_mspct(x, type = "full", validate.names = TRUE, ...)

## S3 method for class 'object_mspct'
join_mspct(x, type = "full", qty.out, validate.names = TRUE, ...)

## S3 method for class 'solute_mspct'
join_mspct(x, type = "full", validate.names = TRUE, ...)
```

Arguments

x	generic_mspct object, or an object of a class derived from `generic_mspct`.

`type`	character Type of join: `"inner"` (default) or `"full"`. See details for more information.
`...`	ignored (possibly used by derived methods).
`col.name`	character, name of the column in the spectra to be preserved, in addition to "w.length".
`validate.names`	logical A flag to enable (default) or disable validation of column names with `make.names`.
`unit.out`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`qty.out`	character Allowed values `"transmittance"`, `"absorptance"`, and `"absorbance"` and in the method for `object_spct`, also `"reflectance"` (.

Value

A `data.frame` with the spectra joined by, possibly interpolated, wavelength, with rows sorted by wavelength (variable `w.length`) and data columns named according to the names of members in `x`, by default made unique and valid.

Methods (by class)

- `join_mspct(default)`:
- `join_mspct(generic_mspct)`:
- `join_mspct(source_mspct)`:
- `join_mspct(response_mspct)`:
- `join_mspct(filter_mspct)`:
- `join_mspct(reflector_mspct)`:
- `join_mspct(object_mspct)`:
- `join_mspct(solute_mspct)`:

Note

Currently only `generic_spct`, `source_mspct`, `response_mspct`, `filter_mspct`, `reflector_mspct`, `object_mspct` and `solute_mspct` classes have this method implemented.

Examples

```
my.mspct <- solute_mspct(list(water = water.spct, pha = phenylalanine.spct))
join_mspct(my.mspct, type = "inner")
join_mspct(my.mspct, type = "full")
```

labels — *Find labels from "waveband" object*

Description

A method specialization that extracts the name and label of objects of class waveband.

Usage

```
## S3 method for class 'waveband'
labels(object, ...)

## S3 method for class 'generic_spct'
labels(object, ...)
```

Arguments

object	an object of class "waveband"
...	not used in current version

Methods (by class)

- labels(generic_spct):

See Also

Other waveband attributes: is_effective(), normalization()

Examples

```
labels(sun.spct)
```

Ler_leaf.spct — *Green Arabidopsis leaf reflectance and transmittance.*

Description

A dataset of total spectral reflectance and total spectral transmittance expressed as fractions of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

Usage

```
Ler_leaf.spct

Ler_leaf_rflt.spct

Ler_leaf_trns.spct

Ler_leaf_trns_i.spct
```

Format

Datasets stored as `object_spct`, `reflector_spct` and `filter_spct` objects, containing transmittance and reflectance data.

An object of class `reflector_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 1750 rows and 2 columns.

An object of class `filter_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 1753 rows and 2 columns.

An object of class `filter_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 2401 rows and 3 columns.

Details

- w.length (nm)
- Rfr (0..1)
- Tfr (0..1)

Note

Measured with a Jaz spectrometer from Ocean Optics (USA) configured with a PX Xenon lamp module and Spectroclip double integrating spheres.

Author(s)

Aphalo, P. J. & Wang, F (unpublished data)

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
Ler_leaf.spct
Ler_leaf_rflt.spct
```

log — *Logarithms and Exponentials*

Description

Logarithms and Exponentials for Spectra. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options

Usage

```
## S3 method for class 'generic_spct'
log(x, base = exp(1))

## S3 method for class 'generic_spct'
log2(x)

## S3 method for class 'generic_spct'
log10(x)

## S3 method for class 'generic_spct'
exp(x)
```

Arguments

x	an object of class "generic_spct"
base	a positive number: the base with respect to which logarithms are computed. Defaults to e=exp(1).

Value

An object of the same class as x.

Note

In most cases a logarithm of an spectral quantity will yield off-range values. For this reason unless x is an object of base class generic_spct, checks will not be passed, resulting in warnings or errors.

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

make_var_labels *Column or variable labels*

Description

Create a named list of character strings describing the variables contained in a spectrum object.

Usage

```
make_var_labels(x, ...)

## Default S3 method:
make_var_labels(x, ...)

## S3 method for class 'source_spct'
make_var_labels(x, ...)

## S3 method for class 'response_spct'
make_var_labels(x, ...)

## S3 method for class 'filter_spct'
make_var_labels(x, ...)

## S3 method for class 'reflector_spct'
make_var_labels(x, ...)

## S3 method for class 'object_spct'
make_var_labels(x, ...)

## S3 method for class 'solute_spct'
make_var_labels(x, ...)

## S3 method for class 'chroma_spct'
make_var_labels(x, ...)

## S3 method for class 'calibration_spct'
make_var_labels(x, ...)

## S3 method for class 'raw_spct'
make_var_labels(x, ...)

## S3 method for class 'cps_spct'
make_var_labels(x, ...)
```

Arguments

x	An object of a class derived from `generic_spct`.

... Currently ignored.

Details

Objects of classes derived from `generic_spct` are used to store different types of spectral data. The data stored in some of the classes needs to be interpreted differently depending on how they were measured or are expressed and this information is stored in attributes of the objects. In other cases, even if consistent across different objects, the units of expression may not be obvious to users. The names of the variables are concise, thus using variable labels makes it possible to make these features visible when exploring the data. The methods provided do not add the labels, only supply the character strings. Variable labels are implemented in packages 'labelled' by setting the `label` attribute in each variable (= column) of a data frame or tibble. This is compatible with the approach used by package 'haven'.

Value

A named list of character strings with one member for each recognized column in `x`. This list can be used to set variable labels with methods from package 'labelled'. However, package 'photobiology' does not natively support variable labels stored in attribute `label`.

Methods (by class)

- `make_var_labels(default)`:
- `make_var_labels(source_spct)`:
- `make_var_labels(response_spct)`:
- `make_var_labels(filter_spct)`:
- `make_var_labels(reflector_spct)`:
- `make_var_labels(object_spct)`:
- `make_var_labels(solute_spct)`:
- `make_var_labels(chroma_spct)`:
- `make_var_labels(calibration_spct)`:
- `make_var_labels(raw_spct)`:
- `make_var_labels(cps_spct)`:

Note

These methods are still under development and the text of the labels may change. Not all classes derived from `generic_spct` are yet supported.

Examples

```
make_var_labels(sun.spct)
# str() prints more compactly than print()
str(make_var_labels(sun.spct))
str(make_var_labels(normalize(sun.spct)))
str(make_var_labels(fscale(sun.spct)))
str(make_var_labels(sun_daily.spct))

str(make_var_labels(polyester.spct))
str(make_var_labels(normalize(polyester.spct)))
str(make_var_labels(fscale(polyester.spct)))

str(make_var_labels(white_led.cps_spct))
str(make_var_labels(white_led.raw_spct))
```

MathFun *Miscellaneous Mathematical Functions*

Description

abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options.

Usage

```
## S3 method for class 'generic_spct'
sqrt(x)

## S3 method for class 'generic_spct'
abs(x)
```

Arguments

x	an object of class "generic_spct"

See Also

Other math operators and functions: ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

merge2object_spct *Merge into object_spct*

Description

Merge a filter_spct with a reflector_spct returning an object_spct object, even if wavelength values are mismatched.

Usage

```
merge2object_spct(
  x,
  y,
  by = "w.length",
  ...,
  w.length.out = x[["w.length"]],
  Tfr.type.out = "total"
)
```

Arguments

`x, y`	a `filter_spct` object and a `reflector_spct` object.
`by`	a vector of shared column names in `x` and `y` to merge on; `by` defaults to `w.length`.
`...`	other arguments passed to `dplyr::inner_join()`.
`w.length.out`	numeric vector of wavelengths to be used for the returned object (nm).
`Tfr.type.out`	character string indicating whether transmittance values in the returned object should be expressed as `"total"` or `"internal"`. This applies only to the case when an `object_spct` is returned.

Value

An `object_spct` is returned as the result of merging a `filter_spct` and a `reflector_spct` object.

Note

If a numeric vector is supplied as argument for `w.length.out`, the two spectra are interpolated to the new wavelength values before merging. The default argument for `w.length.out` is `x[["w.length"]]`.

See Also

`join`

`merge_attributes` *Merge and copy attributes*

Description

Merge attributes from `x` and `y` and copy them to `z`. Methods defined for spectral objects of classes from package 'photobiology'.

Usage

```
merge_attributes(x, y, z, which, which.not, ...)

## Default S3 method:
merge_attributes(x, y, z, which = NULL, which.not = NULL, ...)

## S3 method for class 'generic_spct'
merge_attributes(
  x,
  y,
  z,
  which = NULL,
  which.not = NULL,
  copy.class = FALSE,
  ...
)
```

Arguments

x, y, z	R objects. Objects x and y must be of the same class, z must be an object with a structure valid for this same class.
which	character Names of attributes to copy, if NULL all those relevant according to the class of x are used as default,
which.not	character Names of attributes not to be copied. The names passed here are removed from the list for which, which is most useful when we want to modify the default.
...	not used
copy.class	logical If TRUE class attributes are also copied.

Value

A copy of z with additional attributes set.

Methods (by class)

- `merge_attributes(default)`: Default for generic function
- `merge_attributes(generic_spct)`:

---

minus-.generic_spct *Arithmetic Operators*

Description

Subtraction operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 - e2 = NULL
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

---

mod-.generic_spct *Arithmetic Operators*

Description

Reminder operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 %% e2
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

msmsply *Multi-spct transform methods*

Description

Apply a function or operator to a collection of spectra.

Usage

```
msmsply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)

msdply(
  mspct,
  .fun,
  ...,
  idx = NULL,
  col.names = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

mslply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)

msaply(mspct, .fun, ..., .drop = TRUE, .parallel = FALSE, .paropts = NULL)
```

Arguments

`mspct`	an object of class generic_mspct or a derived class
`.fun`	a function
`...`	other arguments passed to .fun
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`col.names`	character Names to be used for data columns.
`.drop`	should extra dimensions of length 1 in the output be dropped, simplifying the output. Defaults to TRUE

Value

a collection of spectra in the case of `msmsply`, belonging to a different class than `mspct` if `.fun` modifies the class of the member spectra.

a data frame in the case of `msdply`

a list in the case of `mslply`

an vector in the case of `msaply`

`mspct_classes` *Names of multi-spectra classes*

Description

Function that returns a vector containing the names of multi-spectra classes using for collections of spectra.

Usage

```
mspct_classes()
```

Value

A `character` vector of class names.

Examples

```
mspct_classes()
```

`na.omit` *Handle Missing Values in Objects*

Description

These methods are useful for dealing with NAs in e.g., `source_spct`, `response_spct`, `filter_spct` and `reflector_spct`.

Usage

```
## S3 method for class 'generic_spct'
na.omit(object, na.action = "omit", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'response_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'filter_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'reflector_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'object_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'solute_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_spct'
na.exclude(object, na.action = "exclude", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'response_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'filter_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'reflector_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)
```

```
## S3 method for class 'object_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'solute_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)
```

Arguments

`object`	an R object
`na.action`	character One of "omit", "exclude" or "replace".
`fill`	numeric Value used to replace NAs unless NULL, in which case interpolation is attempted.
`target.colnames`	character Vector of names for the target columns to operate upon, if present in `object`.
`...`	further arguments other special methods could require

Details

If `na.omit` removes cases, the row numbers of the cases form the `"na.action"` attribute of the result, of class `"omit"`.

`na.exclude` differs from `na.omit` only in the class of the "na.action" attribute of the result, which is `"exclude"`.

Note

`na.fail` and `na.pass` do not require a specialisation for spectral objects. R's definitions work as expected with no need to override them. We do not define a method `na.replace`, just pass `"replace"` as argument. The current implementation replaces by interpolation only individual NAs which are flanked on both sides by valid data. Runs of multiple NAs con only replaced by a constant value passed through parameter `fill`.

See Also

`na.fail` and `na.action`

Examples

```
my_sun.spct <- sun.spct
my_sun.spct[3, "s.e.irrad"] <- NA
my_sun.spct[5, "s.q.irrad"] <- NA

head(my_sun.spct)

# rows omitted
zo <- na.omit(my_sun.spct)
head(zo)
na.action(zo)

# rows excluded
ze <- na.exclude(my_sun.spct)
head(ze)
na.action(ze)

# data in both rows replaced
zr <- na.omit(my_sun.spct, na.action = "replace")
head(zr)
na.action(zr)
```

---

normalization *Normalization of an R object*

Description

Normalization wavelength $[nm]$ and other normalization metadata of an R object, retrieved from the object's attributes.

Usage

```
normalization(x)

## Default S3 method:
normalization(x)

## S3 method for class 'waveband'
normalization(x)

## S3 method for class 'generic_spct'
normalization(x)

## S3 method for class 'summary_generic_spct'
normalization(x)

## S3 method for class 'generic_mspct'
normalization(x)
```

Arguments

`x`	an R object

Details

In the case of wavebands for spectral weighting functions (`waveband` objects), the normalization wavelength is returned. For spectral objects (`generic_spct` and derived), the normalization descriptor, a list object, is returned. This list contains in addition to the normalization wavelength, the multiplier used and type of normalization applied. These metadata makes it possible to "undo" the normalization and to "update" the normalization after a transformation, such as conversion to a related physical quantity, of the spectral data.

Value

A single numeric value of wavelength $[nm]$ or a list with with members.

Methods (by class)

- `normalization(default)`: Default methods.
- `normalization(waveband)`: Normalization of a `waveband` object.
- `normalization(generic_spct)`: Normalization of a `generic_spct` object.
- `normalization(summary_generic_spct)`: Normalization of a `summary.generic_spct` object.
- `normalization(generic_mspct)`: Normalization of a `generic_mspct` object.

Note

Older versions of the package stored only a subset of the metadata or only a flag to indicate that normalization had been applied. For such objects some or even all fields in the returned list are set to `NA`.

See Also

Other waveband attributes: `is_effective()`, `labels()`

Examples

```
is_normalized(sun.spct)
normalization(sun.spct)
sun_norm.spct <- normalize(sun.spct)
is_normalized(sun_norm.spct)
normalization(sun_norm.spct)

my_wband <- waveband(c(400,700))
is_normalized(my_wband)
normalization(my_wband)
```

`normalize` *Normalize spectral data*

Description

This method returns a spectral object of the same class as the one supplied as argument but with the spectral data normalized to 1.0 at a specific wavelength. When the object contains multiple spectra, the normalisation is applied to each spectrum individually.

Usage

```
normalize(x, ...)

normalise(x, ...)

## Default S3 method:
normalize(x, ...)

## S3 method for class 'source_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  unit.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'response_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  unit.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'filter_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'reflector_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'solute_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'raw_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'cps_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'generic_spct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  col.names,
  keep.scaling = FALSE,
  na.rm = FALSE
)

## S3 method for class 'source_mspct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  unit.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  unit.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)
```

```
## S3 method for class 'reflector_mspct'
normalize(
  x,
  ...,
  range = x,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
normalize(
  x,
  ...,
  range = x,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
normalize(
  x,
  ...,
  range = x,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
normalize(
  x,
  ...,
  range = x,
  norm = "max",
  qty.out = NA,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  col.names,
  keep.scaling = FALSE,
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	An R object
`...`	not used in current version
`range`	An R object on which `range()` returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm) used to set boundaries for search for normalization.
`norm`	numeric Normalization wavelength (nm) or character string "max", or "min" for normalization at the corresponding wavelength, "update" to update the normalization after modifying units of expression, quantity or range but respecting the previously used criterion, "undo" to revert an existing normalization or "skip" to force return of `x` unchanged.
`unit.out`	No longer supported and is ignored with a warning.
`keep.scaling`	logical or numeric Flag to indicate if any existing scaling should be preserved or not. The default, `FALSE`, preserves the behaviour of versions (<= 0.10.9). If numeric, the spectrum is scaled to this value before normalization and marked as not scaled.
`na.rm`	logical indicating whether `NA` values should be stripped before calculating the summary (e.g. "max") used for normalization.
`qty.out`	No longer supported and is ignored with a warning..
`col.names`	character vector containing the names of columns or variables. Columns in `x` matching the names in `col.names` are normalized, other columns are returned unchanged.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

By default normalization is done based on the maximum of the spectral data. It is possible to also do the normalization based on a user-supplied wavelength expressed in nanometres or the minimum. An existing normalization can be updated for a different unit of expression or after a conversion to a related spectral quantity.

By default the function is applied to the whole spectrum, but by passing a range of wavelengths as input, the search, e.g., for the maximum, can be limited to a range of wavelengths of interest instead of the whole spectrum.

In 'photobiology' (>= 0.10.8) detailed information about the normalization is stored in an attribute. In 'photobiology' (>= 0.10.10) applying a new normalization to an already normalized spectrum recomputes the multiplier factors stored in the attributes whenever possible. This ensures that the returned object is identical, except for possible accumulated loss of precision due to floating-point arithmetic, independently of the previous application of a different normalization.

Value

A copy of the object passed as argument to `x` with the values of the spectral quantity rescaled to 1 at the normalization wavelength. If the normalization wavelength is not already present in `x`, it is added by interpolation—i.e. the returned value may be one row longer than `x`. Attributes `normalized` and `normalization` are set to keep a log of the computations applied.

Methods (by class)

- `normalize(default)`: Default for generic function
- `normalize(source_spct)`: Normalize a `source_spct` object.
- `normalize(response_spct)`: Normalize a response spectrum.
- `normalize(filter_spct)`: Normalize a filter spectrum.
- `normalize(reflector_spct)`: Normalize a reflector spectrum.
- `normalize(solute_spct)`: Normalize a solute spectrum.
- `normalize(raw_spct)`: Normalize a raw spectrum.
- `normalize(cps_spct)`: Normalize a cps spectrum.
- `normalize(generic_spct)`: Normalize a raw spectrum.
- `normalize(source_mspct)`: Normalize the members of a source_mspct object.
- `normalize(response_mspct)`: Normalize the members of a response_mspct object.
- `normalize(filter_mspct)`: Normalize the members of a filter_mspct object.
- `normalize(reflector_mspct)`: Normalize the members of a reflector_mspct object.
- `normalize(raw_mspct)`: Normalize the members of a raw_mspct object.
- `normalize(cps_mspct)`: Normalize the members of a cps_mspct object.
- `normalize(solute_mspct)`: Normalize the members of a solute_mspct object.
- `normalize(generic_mspct)`: Normalize the members of a solute_mspct object.

Note

When the spectrum passed as argument to `x` had been previously scaled, in 'photobiology' (<= 0.10.9) the scaling attribute was always removed and no normalization factors returned. In 'photobiology' (>= 0.10.10) scaling information can be preserved by passing `keep.scaling = TRUE`.

By default if `x` contains one or more `NA` values and the normalization is based on a summary quantity, the returned spectrum will contain only `NA` values. If `na.rm == TRUE` then the summary quantity will be calculated after striping `NA` values, and only the values that were `NA` in `x` will be `NA` values in the returned spectrum.

When a numeric value is passed as argument to keep.scaling, the scaling uses `f = "total"` or `f = "mean"` depending on the class of `x`. Rescaling is only occasionally needed.

Method `normalize` is implemented for `solute_spct` objects but as the spectral data stored in them are a description of an intensive property of a substance, normalization is unlikely to useful. To represent solutions of specific concentrations of solutes, `filter_spct` objects should be used instead.

`normalise()` is a synonym for this `normalize()` method.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `setNormalized()`, `setScaled()`

Examples

```
normalize(sun.spct)
normalise(sun.spct) # equivalent

normalize(sun.spct, norm = "max")
normalize(sun.spct, norm = 400)
```

`normalized_diff_ind` *Calculate a normalized difference.*

Description

This method returns a normalized difference index value for an arbitrary pair of wavebands. There are many such indexes in use, such as NDVI (normalized difference vegetation index), NDWI (normalized difference water index), NDMI (normalized difference moisture index), etc., the only difference among then is in the wavebands used.

Usage

```
normalized_diff_ind(spct, w.band.plus, w.band.minus, f, ...)

normalised_diff_ind(spct, w.band.plus, w.band.minus, f, ...)

NDxI(spct, w.band.plus, w.band.minus, f, ...)
```

```
## Default S3 method:
normalized_diff_ind(spct, w.band.plus, w.band.minus, f, ...)

## S3 method for class 'generic_spct'
normalized_diff_ind(spct, w.band.plus, w.band.minus, f, ...)

## S3 method for class 'generic_mspct'
normalized_diff_ind(spct, w.band.plus, w.band.minus, f, ...)
```

Arguments

`spct`	an R object
`w.band.plus, w.band.minus`	waveband objects The wavebands determine the regions of the spectrum used in the calculations.
`f`	function used for integration taking spct as first argument and a list of wavebands as second argument.
`...`	additional arguments passed to f

Details

`f` is most frequently `reflectance`, but also `transmittance`, or even `absorbance`, `response`, `irradiance` or a user-defined function can be used if there is a good reason for it. In every case `spct` should be of the class expected by `f`. When using two wavebands of different widths do consider passing to `f` a suitable `quantity` argument, for example to compare averages rather than integrals. Wavebands can describe weighting functions if desired.

$$\text{NDxI} = \frac{f(s, wb_{\text{plus}}) - f(s, wb_{\text{minus}})}{f(s, wb_{\text{plus}}) + f(s, wb_{\text{minus}})}$$

Value

A named numeric value for the index, or a tibble depending on whether a spectrum or a collection of spectra is passed as first argument. If the wavelength range of `spct` does not fully overlap with both wavebands `NA` is silently returned.

Methods (by class)

- `normalized_diff_ind(default)`: default
- `normalized_diff_ind(generic_spct)`:
- `normalized_diff_ind(generic_mspct)`:

Note

Some NDxI indexes are directly based on satellite instrument data, such as those in the Landsat satellites. To simulate such indexes using spectral reflectande as input, constructors of `waveband` definitions from package 'photobiologyWavebands' can be useful.

`normalised_diff_ind()` is a synonym for `normalized_diff_ind()`.

`NDxI()` is a shorthand for `normalized_diff_ind()`.

See Also

Rfr_normdiff

`normalize_range_arg` *Normalize a range argument into a true numeric range*

Description

Several functions in this package and the suite accept a range argument with a flexible syntax. To ensure that all functions and methods behave in the same way this code has been factored out into a separate function.

Usage

```
normalize_range_arg(arg.range, wl.range, trim = TRUE)
```

Arguments

`arg.range`	a numeric vector of length two, or any other object for which function range() will return a range of wavelengths (nm).
`wl.range`	a numeric vector of length two, or any other object for which function range() will return a range of wavelengths (nm), missing values are not allowed.
`trim`	logical If TRUE the range returned is bound within `wl.range` while if FALSE it can be broader.

Details

The `arg.range` argument can contain NAs which are replaced by the value at the same position in `wl.range`. In addition a NULL argument for `range` is converted into `wl.range`. The `wl.range` is also the limit to which the returned value is trimmed if `trim == TRUE`. The idea is that the value supplied as `wl.range` is the wavelength range of the data.

Value

a numeric vector of length two, guaranteed not to have missing values.

Examples

```
normalize_range_arg(c(NA, 500), range(sun.spct))
normalize_range_arg(c(300, NA), range(sun.spct))
normalize_range_arg(c(100, 5000), range(sun.spct), FALSE)
normalize_range_arg(c(NA, NA), range(sun.spct))
normalize_range_arg(c(NA, NA), sun.spct)
```

oper_spectra *Binary operation on two spectra, even if the wavelengths values differ*

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added.

Usage

```
oper_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE,
  bin.oper = NULL,
  ...
)
```

Arguments

`w.length1`	numeric vector of wavelength (nm)
`w.length2`	numeric vector of wavelength (nm)
`s.irrad1`	a numeric vector of spectral values
`s.irrad2`	a numeric vector of spectral values
`trim`	a character string with value "union" or "intersection"
`na.rm`	a logical value, if TRUE, not the default, NAs in the input are replaced with zeros
`bin.oper`	a function defining a binary operator (for the usual math operators enclose argument in backticks)
`...`	additional arguments (by name) passed to bin.oper

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a dataframe with two numeric variables

`w.length`	A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique vales, sorted in ascending order.
`s.irrad`	A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
head(sun.data)
result.data <-
    with(sun.data,
         oper_spectra(w.length, w.length, s.e.irrad, s.e.irrad, bin.oper=`+`))
head(result.data)
tail(result.data)
my_fun <- function(e1, e2, k) {return((e1 + e2) / k)}
result.data <-
    with(sun.data,
        oper_spectra(w.length, w.length, s.e.irrad, s.e.irrad, bin.oper=my_fun, k=2))
head(result.data)
tail(result.data)
```

`peaks` *Peaks or local maxima*

Description

Function that returns a subset of an R object with observations corresponding to local maxima.

Usage

```
peaks(
  x,
  span,
  global.threshold,
  local.threshold,
  local.reference,
  threshold.range,
  strict,
  na.rm,
  ...
)

## Default S3 method:
peaks(
  x,
  span = NA,
  global.threshold = NA,
  local.threshold = NA,
  local.reference = NA,
  threshold.range = NA,
  strict = NA,
  na.rm = FALSE,
  ...
)

## S3 method for class 'numeric'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'data.frame'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  x.var.name = NULL,
  y.var.name = NULL,
  var.name = y.var.name,
  refine.wl = FALSE,
  method = "spline",
  ...
)
```

```
## S3 method for class 'generic_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'source_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'response_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)
```

```
## S3 method for class 'filter_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'reflector_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'solute_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'cps_spct'
```

```
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_spct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

```
## S3 method for class 'source_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
peaks(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

x	numeric vector.
span	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width span centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. span = NULL extends the span to the whole length of x.
global.threshold	numeric A value belonging to class "AsIs" is interpreted as an absolute minimum height or depth expressed in data units. A bare numeric value (normally between 0.0 and 1.0), is interpreted as relative to threshold.range. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are ignored. A bare negative numeric value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If global.threshold = NULL, no threshold is applied and all peaks returned.
local.threshold	numeric A value belonging to class "AsIs" is interpreted as an absolute minimum height (depth) expressed in data units relative to a within-window computed reference value. A bare numeric value (normally between 0.0 and 1.0),

	is interpreted as expressed in units relative to `threshold.range`. In both cases `local.threshold` sets a *local* height (depth) threshold below which peaks (valleys) are ignored. If `local.threshold = NULL` or if `span` spans the whole of `x`, no threshold is applied.
`local.reference`	character One of `"median"`, `"median.log"`, `"median.sqrt"`, `"farthest"`, `"farthest.log"` or `"farthest.sqrt"`. The reference used to assess the height of the peak, either the minimum/maximum value within the window or the median of all values in the window.
`threshold.range`	numeric vector If of length 2 or a longer vector `range(threshold.range)` is used to scale both thresholds. With `NULL`, the default, `range(x)` is used, and with a vector of length one `range(threshold.range, x)` is used, i.e., the range is expanded.
`strict`	logical flag: if `TRUE`, an element must be strictly greater than all other values in its window to be considered a peak.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for peaks.
`...`	ignored
`var.name, x.var.name, y.var.name`	character Name of column where to look for peaks.
`refine.wl`	logical Flag indicating if peak location should be refined by fitting a function.
`method`	character String with the name of a method. Currently only spline interpolation is implemented.
`unit.out`	character One of "energy" or "photon"
`filter.qty`	character One of "transmittance" or "absorbance"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

As `find_valleys`, `peaks` and `valleys` call `find_peaks` to search for peaks and valleys, this explanation applies to the four functions. It also applies to `stat_peaks` and `stat_valleys`. Function `find_peaks` is a wrapper built onto function `peaks` from **splus2R**, adds support for peak height thresholds and handles `span = NULL` and non-finite (including NA) values differently than `splus2R::peaks`. Instead of giving an error when `na.rm = FALSE` and `x` contains `NA` values, `NA` values are replaced with the smallest finite value in `x`. `span = NULL` is treated as a special case and selects `max(x)`. Passing `strict = TRUE` ensures that non-unique global and within window maxima are ignored, and can result in no peaks being returned.

Two tests make it possible to ignore irrelevant peaks. One test (`global.threshold`) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (`local.threshold`) is available when the window defined by 'span' does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by the argument passed to local.reference. Parameter threshold.range determines how the values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being returned.

The local.threshold argument is used *as is* when local.reference is "median" or "farthest", i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a local.threshold that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with local.reference arguments "median.log", "farthest.log", "median.sqrt", and "farthest.sqrt" as arguments for local.reference.

While functions find_peaks and find_valleys accept as input a numeric vector and return a logical vector, methods peaks and valleys accept as input different R objects, including spectra and collections of spectra and return a subset of the object. These methods are implemented using calls to functions find_peaks, find_valleys and fit_peaks.

Value

A subset of x with rows corresponding to local maxima.

Note

The default for parameter strict is FALSE in functions find_peaks and find_valleys, while the default in peaks is strict = TRUE.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples

```
# default span = 5
peaks(sun.spct)
# global maximum
peaks(sun.spct, span = NULL)
peaks(sun.spct, span = NULL)$w.length
# fitted peak wavelength
peaks(sun.spct, span = NULL, refine.wl = TRUE)
peaks(sun.spct, span = NULL, refine.wl = TRUE)$w.length
# a wider window
peaks(sun.spct, span = 51)
# global threshold relative to the range of s.e.irrad values
peaks(sun.spct, global.threshold = 0.7)
peaks(sun.spct, global.threshold = -0.3)
```

```
# global threshold in actual s.e.irrad values
peaks(sun.spct, global.threshold = 0.7, threshold.range = c(0, 1))
# local threshold  relative to the range of s.e.irrad values
peaks(sun.spct, local.threshold = 0.1)
# local threshold in actual s.e.irrad values
peaks(sun.spct, local.threshold = 0.1, threshold.range = c(0, 1))
# local threshold  relative to the range of s.e.irrad values, using window
# median instead of window minimum
peaks(sun.spct, local.threshold = 0.05, local.reference = "median")
# minimum, the default.
peaks(sun.spct, local.threshold = 0.05, local.reference = "farthest")

peaks(sun.spct)
```

phenylalanine.spct *Molar spectral attenuation coefficient of phenylalanine*

Description

A dataset containing the wavelengths at a 0.25 nm interval and the corresponding attenuation coefficients.

Usage

```
phenylalanine.spct
```

Format

A `solute_spct` object with 1993 rows and 2 variables

Details

- w.length (nm), range 222 to 720 nm.
- K.mole (cm-1/M)

Author(s)

Du et ql. (original data); Scott Prahl (included data).

References

https://omlc.org/spectra/PhotochemCAD/html/073.html

H. Du, R. A. Fuh, J. Li, A. Corkan, J. S. Lindsey, "PhotochemCAD: A computer-aided design and research tool in photochemistry," Photochem. Photobiol., 68, 141-142, 1998.

J. M. Dixon, M. Taniguchi and J. S. Lindsey "PhotochemCAD 2. A refined program with accompanying spectral databases for photochemical calculations", Photochem. Photobiol., 81, 212-213, 2005.

See Also

Other Spectral data examples: A.illuminant.spct, D50.illuminant.spct, D65.illuminant.spct, Ler_leaf.spct, black_body.spct, ccd.spct, clear.spct, filter_cps.mspct, green_leaf.spct, photodiode.spct, sun.spct, sun_daily.spct, sun_evening.spct, two_filters.spct, two_sensors.mspct, water.spct, white_led.source_spct

Examples

```
head(phenylalanine.spct)
summary(phenylalanine.spct)
solute_properties(phenylalanine.spct)
cat(comment(phenylalanine.spct))
```

---

photodiode.spct   *Spectral response of a GaAsP photodiode*

Description

A dataset containing wavelengths at a 1 nm interval and spectral response as $A/(W/nm)$ for GaAsP photodiode type G6262 from Hamamatsu. Data digitized from manufacturer's data sheet. The value at the peak is 0.19 A/W.

Usage

```
photodiode.spct
```

Format

A `response_spct` object with 94 rows and 2 variables

Details

- w.length (nm).
- s.e.response (A/W)

References

Hamamatsu (2011) Datasheet: GaAsP Photodiodes G5645 G5842 G6262. Hamamatsu Photonics KK, Hamamatsu, City. http://www.hamamatsu.com/jp/en/G6262.html. Visited 2017-12-15.

See Also

Other Spectral data examples: A.illuminant.spct, D50.illuminant.spct, D65.illuminant.spct, Ler_leaf.spct, black_body.spct, ccd.spct, clear.spct, filter_cps.mspct, green_leaf.spct, phenylalanine.spct, sun.spct, sun_daily.spct, sun_evening.spct, two_filters.spct, two_sensors.mspct, water.spct, white_led.source_spct

Examples

```
photodiode.spct
```

---

photons_energy_ratio *Photon:energy ratio*

Description

This function gives the photons:energy ratio between for one given waveband of a radiation spectrum.

Usage

```
photons_energy_ratio(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelength (nm).
`s.irrad`	numeric vector of spectral irradiances in $[W\,m^{-2}\,nm^{-1}]$ or $[mol\,s^{-1}\,sm^{-2}\,nm^{-1}]$ as indicated by the argument pased to `unit.in`.
`w.band`	waveband object.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag telling whether to sanity check input data, default is TRUE.
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value giving the ratio moles-photons per Joule.

Note

The default for the `w.band` parameter is a waveband covering the whole range of `w.length`.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
# photons:energy ratio
with(sun.data,
     photons_energy_ratio(w.length, s.e.irrad, new_waveband(400,500)))
# photons:energy ratio for whole spectrum
with(sun.data, photons_energy_ratio(w.length, s.e.irrad))
```

`photon_irradiance` *Photon irradiance*

Description

This function returns the photon irradiance for a given waveband of a radiation spectrum, optionally applies a BSWF.

Usage

```
photon_irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelength $[nm]$.
`s.irrad`	numeric vector of spectral irradiances in $[W\,m^{-2}\,nm^{-1}]$ or $[mol\,s^{-1}\,sm^{-2}\,nm^{-1}]$ as indicated by the argument pased to `unit.in`.
`w.band`	waveband.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag telling whether to sanity check input data, default is `TRUE`.
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.

`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value with no change in scale factor: $[mol\, s^{-1}\, sm^{-2}]$.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data, photon_irradiance(w.length, s.e.irrad))
with(sun.data, photon_irradiance(w.length, s.e.irrad, new_waveband(400,700)))
```

`photon_ratio` *Photo:photon ratio*

Description

This function gives the photon ratio between two given wavebands of a radiation spectrum.

Usage

```
photon_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelength (nm).
`s.irrad`	numeric vector of spectral irradiances in $[W\,m^{-2}\,nm^{-1}]$ or $[mol\,s^{-1}\,sm^{-2}\,nm^{-1}]$ as indicated by the argument pased to `unit.in`.
`w.band.num`	waveband object used to compute the numerator of the ratio.
`w.band.denom`	waveband object used to compute the denominator of the ratio.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag telling whether to sanity check input data, default is TRUE.
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a single numeric value giving the unitless ratio.

Note

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data,
     photon_ratio(w.length,
                  s.e.irrad, new_waveband(400,500), new_waveband(400,700)))
```

plus-.generic_spct *Arithmetic Operators*

Description

Division operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 + e2 = NULL
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct

print.generic_spct *Print spectral objects*

Description

Print methods for objects of spectral classes, including collections of spectra.

Usage

```
## S3 method for class 'generic_spct'
print(x, ..., attr.simplify = TRUE, n = NULL, width = NULL)

## S3 method for class 'generic_mspct'
print(x, ..., attr.simplify = TRUE, n = NULL, width = NULL, n.members = 10)
```

Arguments

x	An object of one of the summary classes for spectra.
...	not used in current version.
attr.simplify	logical If all members share the same attribute value return one copy instead of a data.frame, list or vector.

`n`	Number of rows to show. If NULL, the default, will print all rows if less than option `dplyr.print_max`. Otherwise, will print `dplyr.print_min` rows.
`width`	Width of text output to generate. This defaults to NULL, which means use getOption("width") and only display the columns that fit on one screen. You can also set option(dplyr.width = Inf) to override this default and always print all columns.
`n.members`	numeric Number of members of the collection to print.

Details

This is simply a wrapper on the print method for tibbles, with additional information in the header. Currently, `width` applies only to the table of data.

Objects are printed as is, ignoring the current settings of R options `photobiology.radiation.unit` and `photobiology.filter.qty`.

Value

Returns `x` invisibly.

Functions

- `print(generic_mspct)`:

Examples

```
print(sun.spct)
print(sun.spct, n = 5)

print(q2e(sun.spct, action = "replace"))
print(e2q(sun.spct, action = "replace"))

print(polyester.spct)
print(any2A(polyester.spct))
print(any2Afr(polyester.spct))

print(two_filters.spct)
```

`print.metadata` *Print methods for metadata records*

Description

Print methods for objects of classes used to store different meta data properties in the classes for different types of spectra.

Usage

```
## S3 method for class 'instr_desc'
print(x, ...)

## S3 method for class 'instr_settings'
print(x, ...)

## S3 method for class 'filter_properties'
print(x, ...)

## S3 method for class 'solute_properties'
print(x, ...)
```

Arguments

`x`	An object of one of the summary classes for spectra.
`...`	not used in current version.

Details

These methods print an abbreviated representaion of objects used to store metadata in attributes. They are similar to *records* and formatted printing is useful both on its own and in the print methods for spectra and their summaries.

Examples

```
print(getInstrDesc(sun_evening.spct))
str(getInstrDesc(sun_evening.spct))

print(getInstrSettings(sun_evening.spct))
str(getInstrSettings(sun_evening.spct))

print(filter_properties(polyester.spct))
str(filter_properties(polyester.spct))

print(solute_properties(phenylalanine.spct))
str(solute_properties(phenylalanine.spct))
```

`print.summary_generic_spct` *Print spectral summary*

Description

A function to nicely print objects of classes "summary...spct".

Usage

```
## S3 method for class 'summary_generic_spct'
print(x, ..., attr.simplify = TRUE)

## S3 method for class 'summary_generic_mspct'
print(x, width = NULL, ..., n = NULL)
```

Arguments

`x`	An object of one of the summary classes for spectra
`...`	named arguments passed to the `print()` method for class `"tbl_df"`.
`attr.simplify`	logical If all members share the same attribute value return one copy instead of a data.frame, list or vector.
`width`	integer Width of text output to generate. This defaults to NULL, which means use the width option.
`n`	integer Number of member spectra for which information is printed.

Functions

- `print(summary_generic_mspct)`:

See Also

`formatting`

Examples

```
print(summary(sun.spct))

print(summary(sun_evening.mspct))
```

`print.waveband` *Print a "waveband" object*

Description

A function to more nicely print objects of class "waveband".

Usage

```
## S3 method for class 'waveband'
print(x, ...)
```

Arguments

`x`	an object of class "waveband"
`...`	not used in current version

prod_spectra *Multiply two spectra, even if the wavelengths values differ*

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added.

Usage

```
prod_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

Arguments

`w.length1`	numeric vector of wavelength (nm).
`w.length2`	numeric vector of wavelength (nm).
`s.irrad1`	a numeric vector of spectral values.
`s.irrad2`	a numeric vector of spectral values.
`trim`	a character string with value "union" or "intersection".
`na.rm`	a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a dataframe with two numeric variables

`w.length`	A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique vales, sorted in ascending order.
`s.irrad`	A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors.: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

```
head(sun.data)
square.sun.data <-
  with(sun.data, prod_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(square.sun.data)
tail(square.sun.data)
```

---

pull_sample *Random sample of spectra*

Description

A method to extract a random sample of members from a list, a collection of spectra or a spectrum object containing multiple spectra in long form.

Usage

```
pull_sample(x, size, ...)

## Default S3 method:
pull_sample(x, size, ...)

## S3 method for class 'list'
pull_sample(
  x,
  size = 1,
  replace = FALSE,
  keep.order = TRUE,
  simplify = FALSE,
  ...
)

## S3 method for class 'generic_spct'
pull_sample(x, size = 1, replace = FALSE, keep.order = TRUE, ...)

## S3 method for class 'generic_mspct'
pull_sample(
  x,
  size = 1,
  replace = FALSE,
  recursive = FALSE,
  keep.order = TRUE,
  simplify = FALSE,
  ...
)
```

Arguments

`x`	An R object possibly containing multiple spectra or other components.
`size`	integer The number of spectra to extract, if available.
`...`	currently ignored.
`replace`	logical Sample with or without replacement.
`keep.order`	logical Return the spectra ordered as in `x` or in random order.
`simplify`	logical If `size` = `1`, and `x` is a collection return the spectrum object instead of a collection with it as only member.
`recursive`	logical If `x` is a collection, expand or not member spectra containing multiple spectra in long form into individual members before sampling.

Value

If `x` is an spectrum object, such as a `"filter_spct"` object, the returned object is of the same class but in most cases containing fewer spectra in long form than `x`. If `x` is a collection of spectrum objecta, such as a `"filter_mspct"` object, the returned object is of the same class but in most cases containing fewer member spectra than `x`.

Methods (by class)

- `pull_sample(default)`: Default for generic function
- `pull_sample(list)`: Specialization for generic_spct
- `pull_sample(generic_spct)`: Specialization for generic_spct
- `pull_sample(generic_mspct)`: Specialization for generic_mspct

See Also

See `sample` for the method used for the sampling.

Examples

```
a.list <- as.list(letters)
names(a.list) <- LETTERS
set.seed(12345678)
pull_sample(a.list, size = 8)
pull_sample(a.list, size = 8, keep.order = FALSE)
pull_sample(a.list, size = 8, replace = TRUE)
pull_sample(a.list, size = 8, replace = TRUE, keep.order = FALSE)
pull_sample(a.list, size = 1)
pull_sample(a.list, size = 1, simplify = TRUE)
```

---

q2e *Convert photon-based quantities into energy-based quantities*

Description

Conversion methods for spectral photon irradiance into spectral energy irradiance and for spectral photon response into spectral energy response.

Usage

```
q2e(x, action, byref, ...)

## Default S3 method:
q2e(x, action = "add", byref = FALSE, ...)

## S3 method for class 'source_spct'
q2e(x, action = "add", byref = FALSE, ...)

## S3 method for class 'response_spct'
q2e(x, action = "add", byref = FALSE, ...)

## S3 method for class 'source_mspct'
q2e(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)

## S3 method for class 'response_mspct'
q2e(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
```

Arguments

`x`	an R object.
`action`	a character string, one of "add", or "replace".
`byref`	logical indicating if a new object will be created by reference or a new object returned.
`...`	not used in current version.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

The converted spectral values are added to or replace the existing spectral values depending on the argument passed to parameter `action`. Addition is currently not supported for normalized spectra. If the spectrum has been normalized with a recent version of package 'photobiology' the spectrum will be renormalized after conversion using the same arguments as previously.

Methods (by class)

- `q2e(default)`: Default method
- `q2e(source_spct)`: Method for spectral irradiance
- `q2e(response_spct)`: Method for spectral responsiveness
- `q2e(source_mspct)`: Method for collections of (light) source spectra
- `q2e(response_mspct)`: Method for collections of response spectra

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`

---

qe_ratio *Photon:energy ratio*

Description

This function returns the photon to energy ratio for each waveband of a light source spectrum.

Usage

```
qe_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## Default S3 method:
qe_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## S3 method for class 'source_spct'
qe_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = "[q:e]",
  ...
)
```

```
## S3 method for class 'source_mspct'
qe_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = "[q:e]",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	source_spct.
`w.band`	waveband or list of waveband objects.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical Flag telling whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

The ratio is based on one photon irrandiance and one energy irradiance, both computed for the same waveband.

$$\frac{Q(s, wb)}{I(s, wb)}$$

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

Value

Computed values are ratios between photon irradiance and energy irradiance for a given waveband. A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used, with "[q:e]" prepended. Units are [mol J-1].

Performance

As this method accepts spectra as its input, it computes irradiances before computing the ratios. If you need to compute both ratios and irradiances from several hundreds or thousands of spectra, computing the ratios from previously computed irradiances avoids their repeated computation. A less dramatic, but still important, increase in performance is available when computing in the same function call ratios that share the same denominator.

See Also

Other photon and energy ratio functions: `e_fraction()`, `e_ratio()`, `eq_ratio()`, `q_fraction()`, `q_ratio()`

Examples

```
qe_ratio(sun.spct,
         waveband(c(400,700), wb.name = "White")) # mol J-1
qe_ratio(sun.spct,
         waveband(c(400,700), wb.name = "White"),
         scale.factor = 1e6) # umol J-1
```

`q_fluence` *Photon fluence*

Description

Photon irradiance (i.e. quantum irradiance) for one or more waveband of a light source spectrum.

Usage

```
q_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
q_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
q_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)
```

```
## S3 method for class 'source_mspct'
q_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	a list of `waveband` objects or a `waveband` object
`exposure.time`	lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
`...`	other arguments (possibly ignored).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The exposure.time is copied from the spectrum object to the output as an attribute. Units are as follows: moles of photons per exposure.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_fluence()`, `e_irrad()`, `fluence()`, `irrad()`, `q_irrad()`

Examples

```
library(lubridate)
q_fluence(sun.spct,
         w.band = waveband(c(400,700)),
         exposure.time = lubridate::duration(3, "minutes") )
```

q_fraction *Photon:photon fraction*

Description

This function returns the photon fraction for a given pair of wavebands of a light source spectrum.

Usage

```
q_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
```

```
q_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'source_spct'
q_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'source_mspct'
q_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = ifelse(naming != "none", "[q:q]", ""),
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

spct	an object of class "source_spct".

`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the fraction(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the fraction(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "total"` the fraction is based on two **photon irradiances**, one computed for each waveband.

$$\frac{Q(s, wb_{\mathrm{num}})}{Q(s, wb_{\mathrm{denom}}) + Q(s, wb_{\mathrm{num}})}$$

If the argument is set to `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral photon irradiances**, one computed for each waveband.

$$\frac{\overline{Q_\lambda}(s, wb_{\mathrm{num}})}{\overline{Q_\lambda}(s, wb_{\mathrm{denom}}) + \overline{Q_\lambda}(s, wb_{\mathrm{num}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[q:q]" is appended if `quantity = "total"` and "[q(wl):q(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `q_fraction(default)`: Default for generic function
- `q_fraction(source_spct)`: Method for `source_spct` objects
- `q_fraction(source_mspct)`: Calculates photon:photon from a `source_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other photon and energy ratio functions: `e_fraction()`, `e_ratio()`, `eq_ratio()`, `q_ratio()`, `qe_ratio()`

Examples

```
q_fraction(sun.spct, new_waveband(400,500), new_waveband(400,700))
```

`q_irrad` *Photon irradiance*

Description

Photon irradiance (i.e. quantum irradiance) for one or more wavebands of a light source spectrum.

Usage

```
q_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
q_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
q_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  return.tb = FALSE,
  ...
)

## S3 method for class 'source_mspct'
q_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	a list of `waveband` objects or a `waveband` object.
`quantity`	character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
`time.unit`	character or lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`allow.scaled`	logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
`...`	other arguments (possibly ignored).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`return.tb`	logical Flag forcing a tibble to be always returned, even for a single spectrum as argumnet to `spct`. The default is `FALSE` for backwards compatibility.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach

`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of a `_spct` object containing a single spectrum and `return.tb = FALSE`. The vector has one member one value for each `waveband` passed to parameter `w.band`. In all other cases a `tibble`, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, [W m-2 nm-1] -> [mol s-1 m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [mol d-1 m-2]

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other irradiance functions: `e_fluence()`, `e_irrad()`, `fluence()`, `irrad()`, `q_fluence()`

Examples

```
q_irrad(sun.spct, waveband(c(400,700)))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")
```

q_ratio *Photon:photon ratio*

Description

This function returns the photon ratio for a given pair of wavebands of a light source spectrum.

Usage

```
q_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
q_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'source_spct'
q_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = NULL,
  ...
)
```

```
## S3 method for class 'source_mspct'
q_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "total",
  naming = "short",
  name.tag = "[q:q]",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "source_spct".
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) of the ratio(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach

`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "total"` the ratio is based on two photon irradiances, one computed for each waveband.

$$\frac{Q(s, wb_{\mathrm{num}})}{Q(s, wb_{\mathrm{denom}})}$$

If the argument is set to `quantity = "mean"` or `quantity = "average"` the ratio is based on two mean spectral photon irradiances, one computed for each waveband.

$$\frac{\overline{Q_\lambda}(s, wb_{\mathrm{num}})}{\overline{Q_\lambda}(s, wb_{\mathrm{denom}})}$$

Ratios based on totals and means are numerically identical only if the wavelength expanse of the two wavebands is the same.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[q:q]" is appended if `quantity = "total"` and "[q(wl):q(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Performance

As this method accepts spectra as its input, it computes irradiances before computing the ratios. If you need to compute both ratios and irradiances from several hundreds or thousands of spectra, computing the ratios from previously computed irradiances avoids their repeated computation. A less dramatic, but still important, increase in performance is available when computing in the same function call ratios that share the same denominator.

See Also

Other photon and energy ratio functions: e_fraction(), e_ratio(), eq_ratio(), q_fraction(), qe_ratio()

Examples

```
q_ratio(sun.spct,
        waveband(c(400,500), wb.name = "Blue"),
        waveband(c(400,700), wb.name = "White"))
```

---

q_response *Photon-based photo-response*

Description

This function returns the mean response for a given waveband and a response spectrum.

Usage

```
q_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## Default S3 method:
q_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## S3 method for class 'response_spct'
q_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

## S3 method for class 'response_mspct'
q_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`quantity`	character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
`time.unit`	character or lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.

`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `q_response(default)`: Default method for generic function
- `q_response(response_spct)`: Method for response spectra.
- `q_response(response_mspct)`: Calculates photon (quantum) response from a `response_mspct`

Note

The parameter `use.hinges` controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also

Other response functions: `e_response()`, `response()`

Examples

```
q_response(ccd.spct, new_waveband(200,300))
q_response(photodiode.spct)
```

r4p_pkgs *Packages in R for Photobiology suite*

Description

A dataset containing the names of all the packages in this suite.

Usage

```
r4p_pkgs
```

Format

A `character` `vector`.

Details

A character vector.

Examples

```
r4p_pkgs
```

rbindspct *Row-bind spectra*

Description

A wrapper on `dplyr::rbind_fill` that preserves class and other attributes of spectral objects.

Usage

```
rbindspct(
  l,
  use.names = TRUE,
  fill = TRUE,
  idfactor = TRUE,
  attrs.source = NULL,
  attrs.simplify = FALSE
)
```

Arguments

`l`	A `source_mspct`, `filter_mspct`, `reflector_mspct`, `response_mspct`, `chroma_mspct`, `cps_mspct`, `generic_mspct` object or a list containing `source_spct`, `filter_spct`, `reflector_spct`, `response_spct`, `chroma_spct`, `cps_spct`, or `generic_spct` objects.
`use.names`	logical If `TRUE` items will be bound by matching column names. By default `TRUE` for `rbindspct`. Columns with duplicate names are bound in the order of occurrence, similar to base. When TRUE, at least one item of the input list has to have non-null column names.
`fill`	logical If `TRUE` fills missing columns with NAs. By default `TRUE`. When `TRUE`, `use.names` has also to be `TRUE`, and all items of the input list have to have non-null column names.
`idfactor`	logical or character Generates an index column of `factor` type. Default is (`idfactor=TRUE`) for both lists and `_mspct` objects. If `idfactor=TRUE` then the column is auto named `spct.idx`. Alternatively the column name can be directly provided to `idfactor` as a character string.
`attrs.source`	integer Index into the members of the list from which attributes should be copied. If `NULL`, all attributes are collected into named lists, except that unique comments are pasted.
`attrs.simplify`	logical Flag indicating that when all values of an attribute are equal for all members, the named list will be replaced by a single copy of the value.

Details

Each item of `l` should be a spectrum, including `NULL` (skipped) or an empty object (0 rows). `rbindspc` is most useful when there are a variable number of (potentially many) objects to stack. `rbindspct` always returns at least a `generic_spct` as long as all elements in l are spectra.

Value

An spectral object of a type common to all bound items containing a concatenation of all the items passed in. If the argument 'idfactor' is TRUE, then a factor 'spct.idx' will be added to the returned spectral object.

Note

Note that any additional 'user added' attributes that might exist on individual items of the input list will not be preserved in the result. The attributes used by the `photobiology` package are preserved, and if they are not consistent across the bound spectral objects, a warning is issued.

`dplyr::rbind_fill` is called internally and the result returned is the highest class in the inheritance hierarchy which is common to all elements in the list. If not all members of the list belong to one of the `_spct` classes, an error is triggered. The function sets all data in `source_spct` and `response_spct` objects supplied as arguments into energy-based quantities, and all data in `filter_spct` objects into transmittance before the row binding is done. If any member spectrum is tagged, it is untagged before row binding.

Examples

```
# default, adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct))
spct
class(spct)

# adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'spct.idx' with the names given to the spectra in the list
# supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'ID' with the names given to the spectra in the list
# supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct),
                  idfactor = "ID")
head(spct)
class(spct)
```

`reflectance` *Reflectance*

Description

Function to calculate the mean, total, or other summary of reflectance for spectral data stored in a `reflector_spct` or in an `object_spct`.

Usage

```
reflectance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## Default S3 method:
reflectance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## S3 method for class 'reflector_spct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'object_spct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'reflector_mspct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object
`w.band`	waveband or list of waveband objects or a numeric vector of length two. The

	waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`quantity`	character string One of `"average"` or `"mean"`, `"total"`, `"contribution"`, `"contribution.pc"`, `"relative"` or `"relative.pc"`.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `reflectance(default)`: Default for generic function
- `reflectance(reflector_spct)`: Specialization for reflector_spct
- `reflectance(object_spct)`: Specialization for object_spct
- `reflectance(reflector_mspct)`: Calculates reflectance from a `reflector_mspct`
- `reflectance(object_mspct)`: Calculates reflectance from a `object_mspct`

Note

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

```
reflectance(black_body.spct, waveband(c(400,700)))
reflectance(white_body.spct, waveband(c(400,700)))
```

---

replace_bad_pixs — *Replace bad pixels in a spectrum*

Description

This function replaces data for bad pixels by a local estimate, by either simple interpolation or using the algorithm of Whitaker and Hayes (2018).

Usage

```
replace_bad_pixs(
  x,
  bad.pix.idx = FALSE,
  window.width = 11,
  method = "run.mean",
  na.rm = TRUE
)
```

Arguments

`x`	numeric vector containing spectral data.
`bad.pix.idx`	logical vector or integer. Index into bad pixels in `x`.
`window.width`	integer. The full width of the window used for the running mean.
`method`	character The name of the method: `"run.mean"` is running mean as described in Whitaker and Hayes (2018); `"adj.mean"` is mean of adjacent neighbors (isolated bad pixels only).
`na.rm`	logical Treat `NA` values as additional bad pixels and replace them.

Details

Simple interpolation replaces values of isolated bad pixels by the mean of their two closest neighbors. The running mean approach allows the replacement of short runs of bad pixels by the running mean of neighboring pixels within a window of user-specified width. The first approach works well for spectra from array spectrometers to correct for hot and dead pixels in an instrument. The second approach is most suitable for Raman spectra in which spikes triggered by radiation are wider than a single pixel but usually not more than five pixels wide.

Value

A logical vector of the same length as `x`. Values that are TRUE correspond to local spikes in the data.

Note

In the current implementation NA values are not removed, and if they are in the neighborhood of bad pixels, they will result in the generation of additional NAs during their replacement.

References

Whitaker, D. A.; Hayes, K. (2018) A simple algorithm for despiking Raman spectra. Chemometrics and Intelligent Laboratory Systems, 179, 82-84.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), peaks(), spikes(), valleys(), wls_at_target()

Examples

```
# in a vector
replace_bad_pixs(c(1, 1, 45, 1, 1), bad.pix.idx = 3)

# before replacement
white_led.raw_spct$counts_3[120:125]

# replacing bad pixels at index positions 123 and 1994
with(white_led.raw_spct,
     replace_bad_pixs(counts_3, bad.pix.idx = c(123, 1994)))[120:125]
```

---

response *Integrated response*

Description

Calculate average photon- or energy-based photo-response.

Usage

```
response(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)
```

```
## Default S3 method:
response(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## S3 method for class 'response_spct'
response(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

## S3 method for class 'response_mspct'
response(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object of class "generic_spct".

`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
`unit.out`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`quantity`	character string One of `"average"` or `"mean"`, `"total"`, `"contribution"`, `"contribution.pc"`, `"relative"` or `"relative.pc"`.
`time.unit`	character or lubridate::duration object.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly used by derived methods).
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Whether returned values are expressed in energy-based or photon-based units depends on `unit.out`. By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `response(default)`: Default for generic function
- `response(response_spct)`: Method for response spectra.
- `response(response_mspct)`: Calculates response from a `response_mspct`

Note

The parameter `use.hinges` controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also

Other response functions: `e_response()`, `q_response()`

`Rfr_fraction` *reflectance:reflectance fraction*

Description

This function returns the reflectance fraction for a given pair of wavebands of a reflector spectrum.

Usage

```
Rfr_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Rfr_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'reflector_spct'
Rfr_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'reflector_mspct'
Rfr_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "reflector_spct".
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the fraction(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the fraction(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral reflectance**, one computed for each waveband.

$$\frac{\overline{\mathrm{Rfr}_\lambda}(s, wb_{\mathrm{num}})}{\overline{\mathrm{Rfr}_\lambda}(s, wb_{\mathrm{denom}}) + \overline{\mathrm{Rfr}_\lambda}(s, wb_{\mathrm{num}})}$$

If the argument is set to `quantity = "total"` the fraction is based on two **integrated reflectance**, one computed for each waveband.

$$\frac{\mathrm{Rfr}(s, wb_{\mathrm{num}})}{\mathrm{Rfr}(s, wb_{\mathrm{denom}}) + \mathrm{Rfr}(s, wb_{\mathrm{num}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Rfr:Rfr]" is appended if `quantity = "total"` and "[Rfr(wl):Rfr(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Rfr_fraction(default)`: Default for generic function
- `Rfr_fraction(reflector_spct)`: Method for `reflector_spct` objects
- `Rfr_fraction(reflector_mspct)`: Calculates Rfr:Rfr from a `reflector_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other Reflectance ratio functions: `Rfr_normdiff()`, `Rfr_ratio()`

Examples

```
Rfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"))
Rfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "total")
Rfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "mean")
```

`Rfr_from_n` *Reflectance at a planar boundary*

Description

The reflectance at the planar boundary between two media, or interface, can be computed from the relative refractive index. Reflectance depends on polarization, and the process of reflection can generate polarized light through selective reflection of s and p components. A perfectly flat (i.e., polished) interface creates specular reflection, and this is the case that these functions describe. These function describe a single interface, and for example in a glass pane, a light beam will cross two air-glass interfaces.

Usage

```
Rfr_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5, p_fraction = 0.5)

Rfr_p_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)

Rfr_s_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)
```

Arguments

`angle_deg, angle`	numeric vector Angle of incidence of the light beam, in degrees or radians. If both are supplied, radians take precedence.
`n`	numeric vector, or generic_spct object Relative refractive index. The default 1.5 is suitable for crown glass or acrylic interacting with visible light. n depends on wavelength, more or less strongly depending on the material.
`p_fraction`	numeric in range 0 to 1. Polarization, defaults to 0.5 assuming light that is not polarized.

Details

These functions implement Fresnel's formulae. All parameters accept vectors as arguments. If both n and angle are vectors with length different from one, they should both have the same length. Reflectance depends on polarization, the s and p components need to be computed separately and added up. `Rfr_from_n()` is for non-polarized light, i.e., with equal contribution of the two components.

Value

If `n` is a numeric vector the returned value is a vector of reflectances, while if `n` is a `generic_spct` object the returned value is a `reflector_spct` object.

Examples

```
Rfr_from_n(0:90)
Rfr_from_n(0:90, p_fraction = 1)
Rfr_from_n(0:90, n = 1.333) # water
```

`Rfr_normdiff` *reflectance:reflectance normalised difference*

Description

This function returns the reflectance normalized difference index for a given pair of wavebands of a reflector spectrum.

Usage

```
Rfr_normdiff(
  spct,
  w.band.plus,
  w.band.minus,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Rfr_normdiff(
  spct,
  w.band.plus,
  w.band.minus,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'reflector_spct'
Rfr_normdiff(
  spct,
  w.band.plus = NULL,
  w.band.minus = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'reflector_mspct'
Rfr_normdiff(
  spct,
  w.band.plus = NULL,
  w.band.minus = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
```

```
)
```

Arguments

`spct`	an object of class "reflector_spct".
`w.band.plus, w.band.minus`	waveband object(s) or a list(s) of waveband objects used to compute the additive and subtractive reflectance terms of the normalized difference index.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two values of **mean spectral photon reflectance**, one computed for each waveband.

$$\frac{\overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{plus}) - \overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{minus})}{\overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{plus}) + \overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{minus})}$$

If the argument is set to `quantity = "total"` the fraction is based on two **photon reflectances**, one computed for each waveband.

$$\frac{\mathrm{Rfr}(s, wb_\mathrm{plus}) - \mathrm{Rfr}(s, wb_\mathrm{minus})}{\mathrm{Rfr}(s, wb_\mathrm{plus}) + \mathrm{Rfr}(s, wb_\mathrm{minus})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Rfr:Rfr]" is appended if `quantity = "total"` and "[Rfr(wl):Rfr(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Rfr_normdiff(default)`: Default for generic function
- `Rfr_normdiff(reflector_spct)`: Method for `reflector_spct` objects
- `Rfr_normdiff(reflector_mspct)`: Calculates Rfr:Rfr from a `reflector_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult =T RUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

`normalized_diff_ind`, accepts different summary functions.

Other Reflectance ratio functions: `Rfr_fraction()`, `Rfr_ratio()`

Examples

```
Rfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"))
Rfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "total")
Rfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "mean")
```

`Rfr_ratio` *reflectance:reflectance ratio*

Description

This function returns the reflectance ratio for a given pair of wavebands of a reflector spectrum.

Usage

```
Rfr_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Rfr_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'reflector_spct'
Rfr_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)
```

```
## S3 method for class 'reflector_mspct'
Rfr_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "reflector_spct".
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the ratio(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach

`.paropts` a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral reflectance**, one computed for each waveband.

$$\frac{\overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{num})}{\overline{\mathrm{Rfr}_\lambda}(s, wb_\mathrm{denom}))}$$

If the argument is set to `quantity = "total"` the ratio is based on two **integrated reflectance**, one computed for each waveband.

$$\frac{\mathrm{Rfr}(s, wb_\mathrm{num})}{\mathrm{Rfr}(s, wb_\mathrm{denom})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Rfr:Rfr]" is appended if `quantity = "total"` and "[Rfr(wl):Rfr(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Rfr_ratio(default)`: Default for generic function
- `Rfr_ratio(reflector_spct)`: Method for `reflector_spct` objects
- `Rfr_ratio(reflector_mspct)`: Calculates Rfr:Rfr from a `reflector_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other Reflectance ratio functions: Rfr_fraction(), Rfr_normdiff()

Examples

```
Rfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"))
Rfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"),
          quantity = "total")
Rfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"),
          quantity = "mean")
```

---

rgb_spct *RGB color values*

Description

This function returns the RGB values for a source spectrum.

Usage

```
rgb_spct(spct, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)
```

Arguments

spct	an object of class "source_spct"
sens	a chroma_spct object with variables w.length, x, y, and z, giving the CC or CMF definition (default is the proposed human CMF according to CIE 2006.)
color.name	character string for naming the rgb color definition

Value

A color defined using rgb(). The numeric values of the RGB components can be obtained

See Also

Other color functions: w_length2rgb(), w_length_range2rgb()

Examples

```
rgb_spct(sun.spct)
```

`rmDerivedMspct` *Remove "generic_mspct" and derived class attributes.*

Description

Removes from a spectrum object the class attributes "generic_mspct" and any derived class attribute such as "source_mspct". **This operation is done by reference!**

Usage

```
rmDerivedMspct(x)
```

Arguments

`x`	an R object.

Value

A character vector containing the removed class attribute values. This is different to the behaviour of function `unlist` in base R!

Note

If `x` is an object of any of the multi spectral classes defined in this package, this function changes by reference the multi spectrum object into the underlying list object. Otherwise, it just leaves `x` unchanged. The modified `x` is also returned invisibly.

See Also

Other set and unset 'multi spectral' class functions: `shared_member_class()`

`rmDerivedSpct` *Remove "generic_spct" and derived class attributes.*

Description

Removes from a spectrum object the class attributes `"generic_spct"` and any derived class attribute such as `"source_spct"`. **This operation is done by reference!**

Usage

```
rmDerivedSpct(x, keep.classes = NULL)
```

Arguments

`x`	an R object.
`keep.classes`	character vector Names of classes to keep. Can be used to retain base class `"generic_spct"`.

Details

This function alters `x` itself by reference. If `x` is not a `generic_spct` object, `x` is not modified. This function behaves similarly to `setdiff()` but preserving the original order of the character vector of the S3 class names.

Value

A character vector containing the removed class attribute values. This is different to the behaviour of function `unlist` in base R!

Note

If `x` is an object of any of the spectral classes defined in this package, this function changes by reference the spectrum object into the underlying data.frame object. Otherwise, it just leaves `x` unchanged.

See Also

Other set and unset spectral class functions: `setGenericSpct()`

Examples

```
my.spct <- sun.spct
removed <- rmDerivedSpct(my.spct)
removed
class(sun.spct)
class(my.spct)
```

`round` *Rounding of Numbers*

Description

`ceiling` takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x. \ `floor` takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x. \ `trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward 0. \ `round` rounds the values in its first argument to the specified number of decimal places (default 0). \ `signif` rounds the values in its first argument to the specified number of significant digits. \ The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of `x` and the current value of output options.

Usage

```
## S3 method for class 'generic_spct'
round(x, digits = 0)

## S3 method for class 'generic_spct'
signif(x, digits = 6)

## S3 method for class 'generic_spct'
ceiling(x)

## S3 method for class 'generic_spct'
floor(x)

## S3 method for class 'generic_spct'
trunc(x, ...)
```

Arguments

`x`	an object of class "generic_spct" or a derived class.
`digits`	integer indicating the number of decimal places (round) or significant digits (signif) to be used. Negative values are allowed (see 'Details').
`...`	arguments to be passed to methods.

See Also

Other math operators and functions: `MathFun`, `^.generic_spct()`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`

`select_spct_attributes` *Merge user supplied attribute names with default ones*

Description

Allow users to add and subtract from default attributes in addition to providing a given set of attributes.

Usage

```
select_spct_attributes(attributes, attributes.default = spct_attributes())

spct_attributes(.class = "all", attributes = "*")
```

Arguments

`attributes, attributes.default`	character vector or a list of character vectors.
`.class`	character Name of spectral class.

Details

Vectors of character strings passed as argument to `attributes` are parsed so that if the first member string is `"+"`, the remaining members are added to those in `attributes.default`; if it is `"-"` the remaining members are removed from in `attributes.default`; and if it is `"="` the remaining members replace those in in `attributes.default`. If the first member is none of these three strings, the behaviour is the same as when the first string is `"="`. If `attributes` is `NULL` all the attributes in `attributes.default` are used and if it is `""` no attribute names are returned, `""` has precedence over other member values. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

A character vector of attribute names.

See Also

`get_attributes`

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

`setBSWFUsed` *The "bswf.used" attribute*

Description

Function to set by reference the `"time.unit"` attribute of an existing `source_spct` object, and function to query its value.

Usage

```
setBSWFUsed(x, bswf.used = c("none", "unknown"))

getBSWFUsed(x)
```

Arguments

`x`	a source_spct object.
`bswf.used`	a character string, either `"none"` or the name of a BSWF.

Details

Effective spectral irradiance, describes an estimate of the strength of the radiation towards eliciting a given response, frequently, but not only a biological response. The biological spectral weighting function, BSWF, used, can be for example that of the human eye, or an action spectrum, such as the erythema, or reddening of the human skin, action spectrum.

$$I_{BE}(\lambda) = I(\lambda) \times f_{BE}(\lambda)$$

where, $I_{BE}(\lambda)$ is the biologically effective spectral irradiance, $I(\lambda)$ is the spectral irradiance and $f_{BE}(\lambda)$ is one of many possible BSWF.

When the values stored in a `source_spct` object have been multiplied by those from a curve describing a certain response or effect, the attribute `"time.unit"` is set accordingly to track the transformation applied to the data. When a spectral response data have been directly measured, they should be stored in an object of class `response_spct` as they are expressed in actual response units, not of class `source_spct` expressed in irradiance units, even if weighted. However, when like in the case of spectral illuminance, the aim is technical measure of a light source, class `source_spct` should be used and the BSWF set in the metadata.

This attribute is normally set by the function or operator used to apply the BSWF to spectral irradiance data, or set when the `source_spct` object is created.

Value

`x` or the `character` value stored in `x`.

Note

Function `setBSWFUsed()` alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a `source_spct`, `x` is not modified. The behaviour of this function is 'unusual' in that the default for parameter `bswf.used` is used only if `x` does not already have this attribute set. Function `getBSWFUsed()` returns the value to which the attribute is set as a `character` string and otherwise `NA`.

Examples

```
getBSWFUsed(sun.spct)
```

`setFilterProperties` *Set the "filter.properties" attribute*

Description

Function to set by reference the "filter.properties" attribute of an existing filter_spct object.

Usage

```
setFilterProperties(
  x,
  filter.properties = NULL,
  pass.null = FALSE,
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA_character_,
  verbose = TRUE
)

filter_properties(x) <- value
```

Arguments

`x`	a filter_spct object
`filter.properties`, `value`	a list with fields named `"Rfr.constant"`, `"thickness"` and `"attenuation.mode"`.
`pass.null`	logical If TRUE, the parameters to the next three parameters will be always ignored, otherwise they will be used to build an object of class `"filter.properties"` when the argument passed to parameter `filter.properties` is `NULL`.
`Rfr.constant`	numeric The value of the reflection factor [/1].
`thickness`	numeric The thickness of the material [m].
`attenuation.mode`	character One of `"reflection"`, `"absorption"`, `"absorption.layer"`, `"scattering"`, `"mixed"` or `"stack"`.
`verbose`	logical Flag to enable warning when applied to object of unsuported class.

Details

Storing filter properties allows inter-conversion between internal and total transmittance, as well as computation of transmittance for arbitrary thickness of the material. Whether computations are valid depend on the homogeneity of the material. The parameter `pass.null` makes it possible to remove the attribute.

Value

`x`

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a `filter_spct` object, `x` is not modified.

The values of `attenuation.mode` `"reflection"`, `"absorption"`, `"absorption.layer"` or `"scattering"` should be used when one of these processes is clearly the main one; `"mixed"` is for when multiple modes play a significanr role, i.e., when a simple correction using a single value of `Rfr` across wavelengths is not possible; `"absorption.layer"` is for cases when a thin absorbing layer is deposited on the surface of a transparent support or enclosed between two sheets of glass or other transparent material. Finally `"stack"` is for multiple individual filters piled. If in doubt, set this argument to `NA` to ensure that computation of spectra for a different thickness remains disabled.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes` `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- polyester.spct
filter_properties(my.spct)
filter_properties(my.spct) <- NULL
filter_properties(my.spct)
filter_properties(my.spct, return.null = TRUE)
filter_properties(my.spct) <- list(Rfr.constant = 0.01,
                                   thickness = 125e-6,
                                   attenuation.mode = "absorption")
filter_properties(my.spct)
```

---

setGenericSpct *Convert an R object into a spectrum object.*

Description

Sets the class attribute of a data.frame or an object of a derived class to "generic_spct".

Usage

```
setGenericSpct(x, multiple.wl = 1L, idfactor = NULL)

setCalibrationSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setRawSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setCpsSpct(
  x,
  time.unit = "second",
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setFilterSpct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA_character_,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setSoluteSpct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  name = NA_character_,
  mass = NA_character_,
  formula = NA_character_,
  structure = grDevices::as.raster(matrix()),
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setReflectorSpct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setObjectSpct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setResponseSpct(
  x,
  time.unit = "second",
  response.type = "response",
  multiple.wl = 1L,
  idfactor = NULL
)

setSourceSpct(
  x,
  time.unit = "second",
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setChromaSpct(x, multiple.wl = 1L, idfactor = NULL)
```

Arguments

`x`	data.frame, list or generic_spct and derived classes
`multiple.wl`	numeric Maximum number of repeated `w.length` entries with same value.
`idfactor`	character Name of factor distinguishing multiple spectra when stored longitudinally (required if `mulitple.wl` > 1).
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.
`time.unit`	character string indicating the time unit used for spectral irradiance or exposure (`"second"`, `"day"` or `"exposure"`) or an object of class duration as defined in package lubridate.
`Tfr.type`	character Either "total" or "internal".
`Rfr.constant`	numeric The value of the reflection factor [/1].
`thickness`	numeric The thickness of the material.
`attenuation.mode`	character One of `"reflection"`, `"absorption"` or `"mixed"`.
`K.type`	character A string, either "attenuation", "absorption" or "scattering".
name, `solvent.name`	character The names of the substance and of the solvent. A named character vector, with member names such as "IUPAC" for the authority.
`mass`	numeric The mass in Dalton (Da = g/mol).

`formula`	character The molecular formula.
`structure`	raster A bitmap of the structure.
`ID, solvent.ID`	character The IDs of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChen" for the authority.
`Rfr.type`	character A string, either "total" or "specular".
`response.type`	a character string, either `"response"` or `"action"`.
`bswf.used`	character A string, either `"none"` or the name of a BSWF. (Users seldom need to change the default, as this metadata value is in normal use set by operators or functions that apply a BSWF.)

Details

This method alters `x` itself by reference and in addition returns the modified `x` invisibly. The wavelength values and data are checked for validity and out-of-range values trigger warnings. These checks are done during construction by means of the matching `check_spct` methods, unless checks have been disabled by setting the corresponding option (see `enable_check_spct`).

Value

x

Functions

- `setCalibrationSpct()`: Set class of a an object to "calibration_spct".
- `setRawSpct()`: Set class of a an object to "raw_spct".
- `setCpsSpct()`: Set class of a an object to "cps_spct".
- `setFilterSpct()`: Set class of an object to "filter_spct".
- `setSoluteSpct()`: Set class of an object to "solute_spct".
- `setReflectorSpct()`: Set class of a an object to "reflector_spct".
- `setObjectSpct()`: Set class of an object to "object_spct".
- `setResponseSpct()`: Set class of an object to "response_spct".
- `setSourceSpct()`: Set class of an object to "source_spct".
- `setChromaSpct()`: Set class of an object to "chroma_spct".

Warning!

Not entering metadata when creating an object will limit the available operations!

Note

`"internal"` **transmittance** is defined as the transmittance of the material body itself, while `"total"` transmittance includes the effects of surface reflectance on the amount of light transmitted. For non-diffusing materials like glass an approximate `Rfr.constant` value can be used to inter-convert total and internal transmittance values. Use `NA` if the the mode is not known, or not applicable, e.g., for materials subject to internal scattering. The validity of computations related to thickness of the material or length of the light path depends on the availability and accuracy of the metadata.

Particles in suspension unlike dissolved **solutes** scatter light. Thus two different processes can attenuate light in liquid media: absorption and scattering. Coefficients of attenuation are always based on measurements of internal absorbance or internal transmittance. In practice this is achieved by using as reference pure solvent in a vessel, such as a spectrometer cuvette, called *blank*. The measurement of the blank is done sequentially, before or after the *sample* of interest in single beam spectrophotometers and concurrently in double beam spectrophotometers. `K.type` describes the process of attenuation: `"attenuation"`, `"absorption"` or `"scattering"`, with `"attenuation"` used for cases of mixed modes of attenuation. Set `K.type = NA` if not available or unknown, or not applicable.

`"specular"` **reflectance** is defined as that measured by collecting the light reflected by the surface at the "mirror" of the angle of incidence; i.e., using a probe with a narrow angle of aperture. Usually measured close to normal angle of incidence. `"total"` **reflectance** is defined as that measured by collecting all the light reflected by the surface; i.e., using an integrating sphere. In a mirror, reflectance is mostly specular, while on the white surface of a sheet of paper scattering predominates. In the first case the value for total reflectance is not much more than for specular reflectance, while in the second case the difference is much larger as the "specular" component is much smaller.

See Also

Other set and unset spectral class functions: `rmDerivedSpct()`

Examples

```
my.df <- data.frame(w.length = 300:309, s.e.irrad = rep(100, 10))
is.source_spct(my.df)
setSourceSpct(my.df)
is.source_spct(my.df)
```

`setHowMeasured` *Set the "how.measured" attribute*

Description

Method to set the `"how.measured"` attribute of an R object.

Usage

```
setHowMeasured(x, ...)

how_measured(x) <- value

## Default S3 method:
setHowMeasured(x, how.measured, ...)
```

```
## S3 method for class 'generic_spct'
setHowMeasured(x, how.measured, ...)

## S3 method for class 'summary_generic_spct'
setHowMeasured(x, how.measured, ...)

## S3 method for class 'data.frame'
setHowMeasured(x, how.measured, ...)

## S3 method for class 'generic_mspct'
setHowMeasured(x, how.measured, ...)
```

Arguments

`x`	a R object.
`...`	Allows use of additional arguments in methods for other classes.
`how.measured`, `value`	a list or a character string.

Value

x modified by reference.

Methods (by class)

- `setHowMeasured(default)`: default
- `setHowMeasured(generic_spct)`: generic_spct
- `setHowMeasured(summary_generic_spct)`: summary_generic_spct
- `setHowMeasured(data.frame)`: data.frame
- `setHowMeasured(generic_mspct)`: generic_mspct

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not an object of a supported class, `x` is silently returned unchanged.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`
`setFilterProperties()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
how_measured(my.spct)
how_measured(my.spct) <- "Simulated with a radiation transfer model"
how_measured(my.spct)
how_measured(my.spct) <- NULL
how_measured(my.spct)
```

---

setIdFactor *Set the "idfactor" attribute*

Description

Function to set, rename or unset by reference the "idfactor" attribute of an existing object of class `generic_spct` or an object of a class derived from `generic_spct`.

Usage

```
setIdFactor(x, idfactor)

id_factor(x) <- value
```

Arguments

`x`	a generic_spct object.
`idfactor`, `value`	character The name of a factor identifying multiple spectra stored longitudinally.

Details

If the attribute `idfactor` is already set, and a variable with name equal to the value passed as argument to `idfactor` does not exist in `x`, the currently set variable is renamed and the attribute value updated. If a variable named as the argument passed to `idfactor` exists in `x`, it will be set as id by storing this name in the attribute. If the value passed as argument to `idfactor` is `NULL` the attribute will be unset. If the attribute is not already set and there is no member variable in `x` with a name matching the argument passed to `idfactor`, an error is triggered.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct or an object of a class derived from generic_spct, x is not modified.

See Also

Other idfactor attribute functions: `getIdFactor()`

Examples

```
my.spct <- sun_evening.spct

# inspecting
id_factor(sun.spct) # no idfactor set

id_factor(my.spct)
colnames(my.spct)

# renaming
id_factor(my.spct) <- "time"
getIdFactor(my.spct)
colnames(my.spct)

# removing
setIdFactor(my.spct, NULL)
getIdFactor(my.spct)
colnames(my.spct)
```

---

setInstrDesc — *Set the "instr.desc" attribute*

Description

Function to set by reference the `"instr.desc"` attribute of an existing `generic_spct` or derived-class object, or of a `summary_generic_spct` or derived-class object.

Usage

```
setInstrDesc(x, instr.desc)

instr_descriptor(x) <- value
```

Arguments

`x`	a `generic_spct` object or a `summary_generic_spct` object.
`instr.desc`, `value`	a `list`, `instr_desc` object, or `NULL`.

Details

This function alters x itself by reference and in addition returns x invisibly. If x is not a `generic_spct` object, x is not modified, silently. If `inst.desc` = `NULL` is passed in the call, the attribute `"instr.desc"` is removed. *This function is very rarely called from user code.*

Value

x, with the value of its "instr.desc" attribute set to the value of the argument passed to instr.desc or to value.

Note

The fields to be passed in the list instr.desc in part vary depending on the instrument brand and model.

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes setFilterProperties(), setHowMeasured(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

---

setInstrSettings *Set the "instr.settings" attribute*

Description

Function to set by reference the "what.measured" attribute of a generic_spct, or of a summary_generic_spct object.

Usage

```
setInstrSettings(x, instr.settings)

instr_settings(x) <- value
```

Arguments

x	a generic_spct object or a summary_generic_spct object.
instr.settings, value	a list or a instr_settings object.

Details

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object or a summary_generic_spct object, x is not modified, silently. If inst.desc = NULL is passed in the call, the attribute instr.settings is removed. *This function is very rarely called from user code.*

Value

x

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes setFilterProperties(), setHowMeasured(), setInstrDesc(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

---

setKType *Set the "K.type" attribute*

Description

Function to set by reference the "K.type" attribute of an existing solute_spct object

Usage

```
setKType(x, K.type = c("attenuation", "absorption", "scattering"))
```

Arguments

x	a solute_spct or a summary_solute_spct object.
K.type	character A string, either "attenuation", "absorption" or "scattering".

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a solute_spct object, x is not modified The behaviour of this function is 'unusual' in that the default for parameter K.type is used only if x does not already have this attribute set.

See Also

Other K attribute functions: getKType()

Examples

```
print("missing example")
```

setMultipleWl *Set the "multiple.wl" attribute*

Description

Function to set by reference the `multiple.wl` attribute of an existing `generic_spct` object or an object of a class derived from `generic_spct`.

Usage

```
setMultipleWl(x, multiple.wl = NULL)

multiple_wl(x) <- value
```

Arguments

`x`	a generic_spct object
`multiple.wl, value`	numeric >= 1 If `multiple.wl = NULL`, the default, the value is guessed.

Details

These methods set the attribute `multiple.wl` and if the argument of `multiple.wl` or `value` is `NULL`, they call `findMultipleWl` to obtain a guess. Pathological cases where multiple spectra in long form do not share any wavelength value underestimate the number of spectra, and require an explicit numeric argument. Calling these methods is very rarely needed in user code.

Value

`x`, modified in place by reference. If x is not a `generic_spct` or an object of a class derived from `generic_spct`, x is not modified.

See Also

Other multiple.wl attribute functions: `getMultipleWl()`

Examples

```
my.spct <- sun.spct
setMultipleWl(my.spct) # default is to search x, here my.spct
getMultipleWl(my.spct)

multiple_wl(my.spct) <- 1L # must be a valid value or NULL!
multiple_wl(my.spct)

multiple_wl(my.spct) <- NULL # must be a valid value or NULL!
multiple_wl(my.spct)
```

setNormalized — *Set the "normalized" and "normalization" attributes*

Description

Function to write the "normalized" attribute of an existing generic_spct object.

Usage

```
setNormalized(
  x,
  norm = FALSE,
  norm.type = NA_character_,
  norm.factors = NA_real_,
  norm.cols = NA_character_,
  norm.range = rep(NA_real_, 2),
  verbose = getOption("verbose_as_default", default = FALSE)
)

setNormalised(
  x,
  norm = FALSE,
  norm.type = NA_character_,
  norm.factors = NA_real_,
  norm.cols = NA_character_,
  norm.range = rep(NA_real_, 2),
  verbose = getOption("verbose_as_default", default = FALSE)
)
```

Arguments

`x`	a generic_spct object.
`norm`	numeric (or logical) Normalization wavelength (nanometres).
`norm.type`	character Type of normalization applied.
`norm.factors`	numeric The scaling factor(s) so that dividing the spectral values by this factor reverts the normalization.
`norm.cols`	character The name(s) of the data columns normalized.
`norm.range`	numeric The wavelength range used for normalization (nm).
`verbose`	logical Flag enabling or silencing informative warnings.

Details

This function **is used internally**, although occasionally users may want to use it to "pretend" that spectral data have not been normalized. Use `normalize()` methods to apply a normalization and set the attributes accordingly. Function `setNormalized()` only sets the attributes that store the metadata corresponding to an already applied normalization. Thus a trace of the transformations applied to spectral data is kept, which currently is used to renormalize the spectra when the quantity used for expression is changed with a conversion function. It is also used in other packages like 'ggspectra' when generating automatically axis labels. If `x` is not a `generic_spct` object, `x` is not modified.

Note

Passing a `logical` as argument to `norm` is deprecated but accepted silently for backwards compatibility.

`setNormalised()` is a synonym for this `setNormalized()` method.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setScaled()`

`setResponseType` *Set the "response.type" attribute*

Description

Functions to set by reference the `"response.type"` attribute of an existing `response_spct` object, and to query its value.

Usage

```
setResponseType(x, response.type = c("response", "action"))

getResponseType(x)
```

Arguments

`x`	a response_spct object
`response.type`	a character string, either "response" or "action"

Details

Objects of class `response_spct()` can contain data for a response spectrum or an action spectrum. Response spectra are measured using the same photon (or energy) irradiance at each wavelength. Action spectra are derived from dose response curves at each wavelength, and responsivity at each wavelength is expressed as the reciprocal of the photon fluence required to obtain a fixed level of response. In the case of biological systems the action and response spectra frequently differ in their shape and spectral values. This is a property inherent to a data set and not subject to conversions, thus normally set when a `response_spct` object is created and never modified.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a response_spct object, x is not modified The behaviour of this function is 'unusual' in that the default for parameter `response.type` is used only if x does not already have this attribute set.

Examples

```
my.spct <- ccd.spct
setResponseType(my.spct, "action")
getResponseType(ccd.spct)
getResponseType(sun.spct)
```

`setRfrType` *The "Rfr.type" attribute*

Description

Function to set by reference the `"Rfr.type"` attribute of an existing `reflector_spct` or `object_spct` object, and function to query its current status.

Usage

```
setRfrType(x, Rfr.type = c("total", "specular"))

getRfrType(x)
```

Arguments

`x`	a `reflector_spct` or an `object_spct` object.
`Rfr.type`	character String, either `"total"` or `"specular"`.

Details

Reflectance can be measured by collecting the light reflected out of a surface in all directions, using an integrating sphere, obtaining a quantity called total reflectance. If instead, the reflected light is collected at a narrow angle mirroring the incident angle, only part of the reflected radiation is collected, corresponding to mirror-like reflection, called specular. Thus,

$$\rho = \rho_s + \rho_d$$

where, ρ is total reflectance, and its components, ρ_s, specular reflectance, and ρ_d, diffuse or scattered reflectance. When strong scattering takes place, total reflectance can be much more than the specular component. In most cases ρ_d is not measured directly.

The distinction depends on the measuring procedure, and this information is stored as metadata in an attribute of objects of classes `reflector_spct` or an `object_spct`.

When converting between internal and total transmittance, or computing absorptance by difference based on transmittance and reflectance, only total reflectance can be meaningfully used (if the object does not noticeably scatter light, it may be possible to assume that specular reflectance represents most of the total reflectance.) Consequently, checking the stored value of this attribute is used as a safeguard in these compuations.

This attribute is normally set when the `source_spct` object is created.

Value

`x`, with the modified attribute in the case of `setRfrType()` or the `character` value, `"total"` or `"specular"`, stored in the `"Rfr.type"` attribute of `x` in the case of `getRfrType()`. If `x` is not a `reflector_spct` or an `object_spct` object, `NA` is returned.

Note

Function `setRfrType()` alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a `reflector_spct` or an `object_spct object`, `x` is not modified. The behaviour of this function is 'unusual' in that the default for parameter `Rfr.type` is used only if `x` does not already have this attribute set.

See Also

`reflector_spct` and `object_spct`.

Examples

```
my.spct <- reflector_spct(w.length = 400:409, Rfr = 0.1)
getRfrType(my.spct)
setRfrType(my.spct, "specular")
getRfrType(my.spct)
```

`setScaled` *Set the "scaled" attribute*

Description

Function to write the "scaled" attribute of an existing generic_spct object.

Usage

```
setScaled(x, ...)

## Default S3 method:
setScaled(x, ...)
```

```
## S3 method for class 'generic_spct'
setScaled(x, ..., scaled = FALSE)

## S3 method for class 'summary_generic_spct'
setScaled(x, ..., scaled = FALSE)

## S3 method for class 'generic_mspct'
setScaled(x, ..., scaled = FALSE)
```

Arguments

x	a generic_spct object.
...	currently ignored.
scaled	logical with FALSE meaning that values are expressed in absolute physical units and TRUE meaning that relative units are used. If NULL the attribute is not modified.

Value

a new object of the same class as x.

a new object of the same class as x.

a new object of the same class as x.

a new object of the same class as x.

Methods (by class)

- `setScaled(default)`: Default for generic function
- `setScaled(generic_spct)`: Specialization for generic_spct
- `setScaled(summary_generic_spct)`: Specialization for summary_generic_spct
- `setScaled(generic_mspct)`: Specialization for generic_mspct

Note

if x is not a `generic_spct` object, x is not modified.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setNormalized()`

setSoluteProperties *Set the "solute.properties" attribute*

Description

Function to set by reference the "solute.properties" attribute of an existing solute_spct object.

Usage

```
setSoluteProperties(
  x,
  solute.properties = NULL,
  pass.null = FALSE,
  mass = NA_real_,
  formula = NULL,
  structure = grDevices::as.raster(matrix()),
  name = NA_character_,
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
  verbose = TRUE
)

solute_properties(x) <- value
```

Arguments

x	solute_spct A spectrum of coefficients of attenuation.
solute.properties, value	a list with fields named "mass", "formula", "structure", "name" and "ID".
pass.null	logical If TRUE, the parameters to the next three parameters will be always ignored, otherwise they will be used to build an object of class "solute.properties" when the argument to solute.properties is NULL.
mass	numeric The mass in Dalton [$Da = g\,mol^{-1}$].
formula	character The molecular formula.
structure	raster A bitmap of the structure.
name, solvent.name	character The name of the substance and the name of the solvent. A named character vector, with member names such as "IUPAC" for the authority.
ID, solvent.ID	character The names of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChen" for the authority.
verbose	logical Flag to enable warning when applied to object of unsuported class.

Details

Storing solute properties allows inter-conversion between bases of expression, and ensures the unambiguous identification of the substances to which the spectral data refer. These properties make it possible to compute `filter_spct` objects for solutions of the solute, i.e., absorption spectra of liquid filters. The parameter `pass.null` makes it possible to remove the attribute. The solvent used for the determination of the attenuation coefficient is important metadata as the solvent can alter the spectral ansorption properties of the solute.

Value

x

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a filter_spct object, `x` is not modified.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`
`setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
solute.properties <-
  list(formula = c(text = "H2O", html = "H<sub>2</sub>", TeX = "$H_2O$"),
       name = c("water", IUPAC = "oxidane"),
       structure = grDevices::as.raster(matrix()),
       mass = 18.015, # Da
       ID = c(ChemSpider = "917", CID = "962"),
       solvent.name = NA_character_,
       solvent.ID = NA_character_)
my.spct <- solute_spct()
solute_properties(my.spct) <- solute.properties
solute_properties(my.spct)
solute_properties(my.spct) <- NULL
solute_properties(my.spct)
solute_properties(my.spct, return.null = TRUE)
solute_properties(my.spct)
```

setTfrType *The "Tfr.type" attribute*

Description

Function to set by reference the "Tfr.type" attribute of an existing filter_spct or object_spct object, and function to query its current status.

Usage

```
setTfrType(x, Tfr.type = c("total", "internal"))

getTfrType(x)
```

Arguments

x	a filter_spct or an object_spct object.
Tfr.type	character string, either "total" or "internal".

Details

Transmittance, T or τ, has two different definitions that differ in how reflectance is taken into account: "total" transmittance and "internal" transmittance. They are both in widespread use, and rather frequently the interconversion is approximate or even not possible.

$$T = \frac{I_z}{I_0}$$

$$\tau = \frac{I_z}{I_0 - \rho}$$

where T is total transmittance and τ is internal transmittance; I_0 is the radiant power incident on an object and I_z is the radiant power at depth z, in most cases measured below the non-illuminated side of the object, and ρ is the total reflectance at the illuminated surface.

The transmittance of an object as a whole depends on the length of the light path within the object and reflectance on the angle of incidence of the light on the surface. When the light beam is near-normal to the surface, both quantities are at their minimum.

Thus, the interconversion of total spectral transmittance, $T(\lambda)$, into internal spectral transmittance, $\tau(\lambda)$, is strictly possible only if the spectral reflectance $\rho(\lambda)$ is known. In practice, the spectral reflectance is approximated by a constant value that is assumed independent of wavelength.

Objects of class object_spct contain spectral data for both spectral transmittance and spectral reflectance or spectral absorptance, making conversion possible. Objects of class filter_spct do not contain spectral reflectance data, but may have a known approximate value for a reflectance constant, but this is frequently not the case.

The type of transmittance data stored in an object of these classes is recorded as metadata in attribute Tfr.Type. The functions described here set and query this attribute. Contrary to directly accessing the attribute, the query function consistently returns NA both when the attribute is set to NA and when the attribute has not been set, as can be the case of objects created with early versions of the package.

Absorptance, α, and absorbance, A, are normally given as "internal", and this is the assumption in this package. However, as in some cases strict enforcement would prevent conversions, this is not strictly enforced. (IUPAC, recommends use of the name *attenuance* (formerly *extinction*) instead of *absorbance* when light attenuation involves processes other than pure absorption, such as scattering and luminescence.)

$$1 = \alpha + \rho + \tau$$

$$A_{10} = \log_{10} \frac{1}{\alpha} = -\log_{10} \alpha$$

When a solvent-only *blank* is used when measuring the absorbance of a solution, the absorbance is not only "internal" to the solution (discounting reflections at the cuvette boundaries) but also discounts the effect of the solvent itself. When measuring solid samples, like a sheet of glass, in most cases a blank is not available.

For semitransparent objects like glass, it is important to take into account that reflections occur at each interface between substances with different refractive index.

This attribute is normally set when the source_spct object is created. But convertTfrType() updates it when it changes due to a conversion.

Value

x, with the modified attribute in the case of setTfrType() or the character value, "total" or internal, stored in the "Tfr.type" attribute of x in the case of getTfrType(). If x is not a filter_spct or an object_spct object, NA is returned.

Note

Function setTfrType() alters x itself by reference and in addition returns x invisibly. If x is not a filter_spct or an object_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter Tfr.type is used only if x does not already have this attribute set.

See Also

convertTfrType, filter_spct, and object_spct.

Examples

```
my.spct <- polyester.spct
getTfrType(my.spct)
setTfrType(my.spct, "internal")
getTfrType(my.spct)
```

`setTimeUnit` *Set the "time.unit" attribute of an existing source_spct object*

Description

Function to set by reference the "time.unit" attribute

Usage

```
setTimeUnit(
  x,
  time.unit = c("second", "hour", "day", "exposure", "none"),
  override.ok = FALSE
)
```

Arguments

`x`	a source_spct object
`time.unit`	character string indicating the time unit used for spectral irradiance or exposure ("second" , "day" or "exposure") or an object of class duration as defined in package lubridate.
`override.ok`	logical Flag that can be used to silence warning when overwriting an existing attribute value (used internally)

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a source_spct or response_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter `time.unit` is used only if `x` does not already have this attribute set. `time.unit = "hour"` is currently not fully supported.

See Also

Other time attribute functions: `checkTimeUnit()`, `convertThickness()`, `convertTimeUnit()`, `getTimeUnit()`

Examples

```
my.spct <- sun.spct
setTimeUnit(my.spct, time.unit = "second")
setTimeUnit(my.spct, time.unit = lubridate::duration(1, "seconds"))
```

`setWhatMeasured` *Set the "what.measured" attribute*

Description

Method to set by reference the "`what.measured`" attribute of an R object.

Usage

```
setWhatMeasured(x, ...)

what_measured(x) <- value

## Default S3 method:
setWhatMeasured(x, what.measured, ...)

## S3 method for class 'generic_spct'
setWhatMeasured(x, what.measured, ...)

## S3 method for class 'summary_generic_spct'
setWhatMeasured(x, what.measured, ...)

## S3 method for class 'data.frame'
setWhatMeasured(x, what.measured, ...)

## S3 method for class 'generic_mspct'
setWhatMeasured(x, what.measured, ...)
```

Arguments

`x`	an R object.
`...`	Allows use of additional arguments in methods for other classes.
`what.measured`, `value`	a list

Details

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` does not belong to one of the supported classes, `x` is not modified.

Value

x

Methods (by class)

- `setWhatMeasured(default)`: default
- `setWhatMeasured(generic_spct)`: generic_spct
- `setWhatMeasured(summary_generic_spct)`: summary_generic_spct
- `setWhatMeasured(data.frame)`: data.frame
- `setWhatMeasured(generic_mspct)`: generic_mspct

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
what_measured(my.spct)
what_measured(my.spct) <- "Sun"
what_measured(my.spct)
what_measured(my.spct) <- NULL
what_measured(my.spct)
```

`setWhenMeasured` *Set the "when.measured" attribute*

Description

Method to set by reference the `"when.measured"` attribute of an R object.

Usage

```
setWhenMeasured(x, when.measured, ...)

when_measured(x) <- value

## Default S3 method:
setWhenMeasured(x, when.measured, ...)

## S3 method for class 'generic_spct'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)

## S3 method for class 'summary_generic_spct'
```

```
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)

## S3 method for class 'data.frame'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)

## S3 method for class 'generic_mspct'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)
```

Arguments

`x`	an R object
`when.measured, value`	POSIXct to add as attribute, or a list of POSIXct.
`...`	Allows use of additional arguments in methods for other classes.

Details

This method alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not an object of a supported class, `x` is not modified. If the arguments to `"when.measured"` or `value` are not a `POSIXct` object or `NULL` an error is triggered. A `POSIXct` describes an instant in time (date plus time-of-day plus time zone).

Be aware that `lubridate::ymd()` returns an incompatible `Date` object while `lubridate::ymd_h()`, `lubridate::ymd_hm()` and `lubridate::ymd_hms()` and similar functions return objects of class `POSIXct` acceptable as arguments for parameter `when.measured`.

Value

`x`, with its `"when.measured"` set.

Methods (by class)

- `setWhenMeasured(default)`: default
- `setWhenMeasured(generic_spct)`: generic_spct
- `setWhenMeasured(summary_generic_spct)`: summary_generic_spct
- `setWhenMeasured(data.frame)`: data.frame
- `setWhenMeasured(generic_mspct)`: generic_mspct

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`
`setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
when_measured(my.spct)
when_measured(my.spct) <- lubridate::ymd_hms("2020-01-01 08:00:00")
when_measured(my.spct)
when_measured(my.spct) <- NULL
when_measured(my.spct)
```

`setWhereMeasured` *Set the "where.measured" attribute*

Description

Method to set by reference the `"where.measured"` attribute of an R object.

Usage

```
setWhereMeasured(x, where.measured, lat, lon, address, ...)

where_measured(x) <- value

## Default S3 method:
setWhereMeasured(x, where.measured, lat, lon, address, ...)

## S3 method for class 'generic_spct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)

## S3 method for class 'summary_generic_spct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)

## S3 method for class 'data.frame'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)

## S3 method for class 'generic_mspct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)
```

Arguments

`x`	an R object
`where.measured, value`	A one row `data.frame` with the same format as returned by function geocode from package 'ggmap' for a location search.
`lat`	numeric Latitude in decimal degrees North.
`lon`	numeric Longitude in decimal degrees West.
`address`	character Human readable address.
`...`	Allows use of additional arguments in methods for other classes.

Value

x, with the "where.measured" attribute set.

Methods (by class)

- `setWhereMeasured(default)`: default
- `setWhereMeasured(generic_spct)`: generic_spct
- `setWhereMeasured(summary_generic_spct)`: summary_generic_spct
- `setWhereMeasured(data.frame)`: data.frame
- `setWhereMeasured(generic_mspct)`: generic_mspct

Note

This method alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not an object of a supported class, `x` is not modified. If the argument to `where.measured` is not a `POSIXct` object or `NULL` an error is triggered. A `POSIXct` describes an instant in time (date plus time-of-day plus time zone). As with `attr()` passing `NULL` as argument for parameter `where.measured` unsets the attribute.

Method for collections of spectra recycles the location information only if it is a one row `data.frame`.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`
`setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```
my.spct <- sun.spct
where_measured(my.spct)
where_measured(my.spct) <- data.frame(lon = 0, lat = -60)
where_measured(my.spct)
where_measured(my.spct) <- NULL
where_measured(my.spct)
```

`shared_member_class` *Classes common to all collection members.*

Description

Finds the set intersection among the class attributes of all collection member as a target set of class names.

Usage

```
shared_member_class(l, target.set = spct_classes())
```

Arguments

`l`	a list or a generic_mspct object or of a derived class.
`target.set`	character The target set of classes within which to search for classes common to all members.

Value

A character vector containing the class attribute values.

See Also

Other set and unset 'multi spectral' class functions: `rmDerivedMspct()`

`sign` *Sign*

Description

`sign` returns a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).

Usage

```
## S3 method for class 'generic_spct'
sign(x)
```

Arguments

`x`	an object of class "generic_spct"

See Also

Other math operators and functions: `MathFun`, `^.generic_spct()`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `slash-.generic_spct`, `times-.generic_spct`

slash-.generic_spct *Arithmetic Operators*

Description

Division operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 / e2
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), times-.generic_spct

smooth_spct *Smooth a spectrum*

Description

These functions implement one original methods and acts as a wrapper for other common R smoothing functions. The advantage of using this function for smoothing spectral objects is that it simplifies the user interface and sets, when needed, defaults suitable for spectral data.

Usage

```
smooth_spct(x, method, strength, wl.range, ...)

## Default S3 method:
smooth_spct(x, method, strength, wl.range, ...)

## S3 method for class 'source_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'filter_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'reflector_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'solute_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'response_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)

## S3 method for class 'generic_mspct'
smooth_spct(
  x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
  na.rm = FALSE,
  ...
)
```

Arguments

`x`	an R object.
`method`	a character string "custom", "lowess", "supsmu" or "skip"..
`strength`	numeric value to adjust the degree of smoothing. Ignored if method-specific parameters are passed through `...`.
`wl.range`	any R object on which applying the method `range()` yields a vector of two numeric values, describing a range of wavelengths (nm) within which spectral data is to be smoothed. `NA` is interpreted as the min or max value of `x[[w.length]]`.
`...`	other parameters passed to the underlying smoothing functions.
`na.rm`	logical A flag indicating whether NA values should be stripped before the computation proceeds.

Value

A copy of `x` with spectral data values replaced by smoothed ones.

Methods (by class)

- `smooth_spct(default)`: Default for generic function
- `smooth_spct(source_spct)`: Smooth a source spectrum
- `smooth_spct(filter_spct)`: Smooth a filter spectrum
- `smooth_spct(reflector_spct)`: Smooth a reflector spectrum
- `smooth_spct(solute_spct)`: Smooth a solute attenuation spectrum
- `smooth_spct(response_spct)`: Smooth a response spectrum
- `smooth_spct(cps_spct)`: Smooth a counts per second spectrum
- `smooth_spct(generic_mspct)`:

Note

Method "custom" is our home-brewed method which applies strong smoothing to low signal regions of the spectral data, and weaker or no smoothing to the high signal areas. Values very close to zero are set to zero with a limit which depends on the local variation. This method is an ad-hock method suitable for smoothing spectral data obtained with spectrometers. In the cased of methods "lowess" and "supsmu" the current function behaves like a wrapper of the functions of the same names from base R. Method "skip" returns x unchanged.

Examples

```
my.spct <- clip_wl(sun.spct, c(400, 500))
smooth_spct(my.spct)
smooth_spct(my.spct, method = "custom", strength = 1)
smooth_spct(my.spct, method = "custom", strength = 4)
smooth_spct(my.spct, method = "supsmu", strength = 4)
```

---

source_spct *Spectral-object constructors*

Description

These constructor functions can be used to create spectral objects derived from `generic_spct`. They take as arguments numeric vectors for the wavelengths and spectral data, and numeric, character, and logical values for metadata attributes to be saved to the objects created and options controlling the creation process.

Usage

```
source_spct(
  w.length = NULL,
  s.e.irrad = NULL,
  s.q.irrad = NULL,
  ...,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

calibration_spct(
  w.length = NULL,
  irrad.mult = NA_real_,
  ...,
  comment = NULL,
  instr.desc = NA,
  multiple.wl = 1L,
  idfactor = NULL
)

raw_spct(
  w.length = NULL,
  counts = NA_real_,
  ...,
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL
)

cps_spct(
  w.length = NULL,
  cps = NA_real_,
  ...,
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL
)

generic_spct(
  w.length = NULL,
  ...,
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL
)

response_spct(
  w.length = NULL,
  s.e.response = NULL,
  s.q.response = NULL,
  ...,
  time.unit = c("second", "day", "exposure"),
  response.type = c("response", "action"),
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL
)

filter_spct(
  w.length = NULL,
  Tfr = NULL,
  Tpc = NULL,
  Afr = NULL,
  A = NULL,
  ...,
  Tfr.type = c("total", "internal"),
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA,
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

reflector_spct(
  w.length = NULL,
  Rfr = NULL,
  Rpc = NULL,
  ...,
  Rfr.type = c("total", "specular"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

solute_spct(
  w.length = NULL,
  K.mole = NULL,
  K.mass = NULL,
  attenuation.XS = NULL,
  ...,
  log.base = 10,
  K.type = c("attenuation", "absorption", "scattering"),
  name = NA_character_,
  mass = NA_character_,
  formula = NULL,
  structure = grDevices::as.raster(matrix()),
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)
```

```
object_spct(
  w.length = NULL,
  Rfr = NULL,
  Tfr = NULL,
  Afr = NULL,
  ...,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

chroma_spct(
  w.length = NULL,
  x,
  y,
  z,
  ...,
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)
```

Arguments

`w.length`	numeric vector with wavelengths in nanometres $[nm]$.
`s.e.irrad`	numeric vector with spectral energy irradiance in $[W\,m^{-2}\,nm^{-1}]$ or $[J\,d^{-1}\,m^{-2}\,nm^{-1}]$.
`s.q.irrad`	numeric A vector with spectral photon irradiance in $[mol\,s^{-1}\,m^{-2}\,nm^{-1}]$ or $[mol\,d^{-1}\,m^{-2}\,nm^{-1}]$.
`...`	other arguments passed to `tibble()` such as vectors or factors to be added as additional columns.
`time.unit`	character string indicating the time unit used for spectral irradiance or exposure (`"second"`, `"day"` or `"exposure"`) or an object of class duration as defined in package lubridate.
`bswf.used`	character A string indicating the BSWF used, if any, for spectral effective irradiance or exposure (`"none"` or the name of the BSWF).
`comment`	character A string to be added as a comment attribute to the object created.
`strict.range`	logical Flag indicating whether off-range values result in an error instead of a warning.
`multiple.wl`	numeric Maximum number of repeated `w.length` entries with same value. (As with multiple spectra stored in long from).
`idfactor`	character Name of factor distinguishing multiple spectra when stored longitudinally (required if `multiple.wl` > 1).

`irrad.mult`	numeric vector with multipliers for each detector pixel expressed in units of $W\,m^{-2}\,nm^{-1}\,n^{-1}\,s$, where $n\,s^{-1}$ are detector counts per second.
`instr.desc`	a list describing the spectrometer used to acquire the data.
`counts`	numeric vector with raw counts expressed per scan.
`instr.settings`	a list describing the settings used to acquire the data.
`cps`	numeric vector with linearized raw counts expressed per second [$n\,s^{-1}$]
`s.e.response`	numeric vector with a biological, chemical or physical response expressed per unit spectral energy irradiance [$W\,m^{-2}\,nm^{-1}$ or $J\,d^{-1}\,m^{-2}\,nm^{-1}$].
`s.q.response`	numeric vector with a biological, chemical or physical response expressed per unit spectral photon irradiance in [$mol\,s^{-1}\,m^{-2}\,nm^{-1}$ or $mol\,d^{-1}\,m^{-2}\,nm^{-1}$].
`response.type`	a character string, either `"response"` or `"action"`.
`Tfr`	numeric vector with spectral transmittance as fraction of one [/1].
`Tpc`	numeric vector with spectral transmittance as percent values
`Afr`	numeric vector of absorptance as fraction of one [/1].
`A`	numeric vector of absorbance values (log_{10}-base a.u.)
`Tfr.type`	character string indicating whether transmittance and absorptance values are `"total"` or `"internal"` values
`Rfr.constant`	numeric The value of the reflection factor [/1].
`thickness`	numeric The thickness of the material.
`attenuation.mode`	character One of `"reflection"`, `"absorption"` or `"mixed"`.
`Rfr`	numeric vector with spectral reflectance as fraction of one [/1].
`Rpc`	numeric vector with spectral reflectance as percent values.
`Rfr.type`	character A string, either `"total"` or `"specular"`.
`K.mole`	numeric vector with molar attenuation coefficient in SI units [$m^2\,mol^-1$].
`K.mass`	numeric vector with mass attenuation coefficient in SI units [$m^2\,g^-1$].
`attenuation.XS`	numeric vector with attenuation cross section values (Converted during object construction into `K.mole`.)
`log.base`	numeric Normally one of `e` or `10`. Data are stored always on base 10 corresponding to decadal absorbance as used in chemistry.
`K.type`	character A string, either `"attenuation"`, `"absorption"` or `"scattering"`.
`name`, `solvent.name`	character The names of the substance and of the solvent. A named character vector, with member names such as "IUPAC" for the authority.
`mass`	numeric The molar mass in Dalton [Da] ($Da = g\,mol^{-1}$).
`formula`	character The molecular formula.
`structure`	raster A bitmap of the structure.
`ID`, `solvent.ID`	character The ID of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChem" for the authority.
`x`, `y`, `z`	numeric colour coordinates

Details

Constructors can be used to create spectral objects from spectral quantities expressed on a single base or unit. Some of the functions have different formal parameters accepting a quantity expressed in different units, however, an argument can be passed to only one of these formal parameters in a given call. The constructors `object_spct()` and `chroma_spct()` require arguments to be passed for multiple but distinct spectral quantities.

Value

A object of class `generic_spct` or a class derived from it, depending on the function used. In other words an object of a class with the same name as the constructor function.

Warning for filter_spct!

Not entering metadata when creating an object will limit the available operations! While "internal" transmittance is defined as the transmittance of the material body itself, "total" transmittance includes the effects of surface reflectance on the amount of light transmitted. For non-diffusing materials like glass an approximate `Rfr.constant` value can be used to convert "total" into "internal" transmittance values and vice versa. Use `NA` if not known, or not applicable, e.g., for materials subject to internal scattering.

Warning for solute_spct!

You should always set the base for logarithms to match that on which the absorbance data are expressed. Failing to do this will result in bad data and all further computation will be wrong. Not entering metadata when creating an object will limit the available operations! Mass should be indicated in daltons or $g\,mol^{-1}$. The SI unit of molar attenuation coefficient is the square metre per mole ($m^2\,mol^1$), but in practice, quantities are usually expressed in terms of $M^{-1}\,cm^{-1}$ or $l\,mol^{-1}\,cm^{-1}$ (the latter two units are both equal to 0.1 $m^2\,mol^{-1}$ and quantities expressed in them need to be divided by 10 when passed as arguments to `K.mole`.).

See Also

`setFilterProperties`

`setSoluteProperties`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.solute_spct()`, `as.source_spct()`

`spct_attr2tb` *Copy attributes into a tibble*

Description

Method returning attributes of an object of class generic_spct or derived, or of class waveband. Only attributes defined and/or set by package 'photobiology' for objects of the corresponding class are returned.

Usage

```
spct_attr2tb(
  x,
  which = c("-", "names", "row.names", "spct.tags", "spct.version", "comment"),
  ...
)
```

Arguments

x	a generic_spct object.
which	character vector Names of attributes to retrieve.
...	currently ignored

Value

A tibble with the values stored in the attributes whose names were selected through the argument to `which` if present in `x`.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`, `getInstrDesc()`, `getInstrSettings()`, `getSoluteProperties()`, `getWhatMeasured()`, `getWhenMeasured()`, `getWhereMeasured()`, `get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes`, `setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setSoluteProperties()`, `setWhatMeasured()`, `setWhenMeasured()`, `setWhereMeasured()`, `spct_metadata()`, `subset_attributes()`, `trimInstrDesc()`, `trimInstrSettings()`

`spct_classes` *Function returning a vector containing the names of spectra classes.*

Description

Function returning a vector containing the names of spectra classes.

Usage

```
spct_classes()
```

Value

A `character` vector of class names.

Examples

```
spct_classes()
```

spct_metadata *Access metadata*

Description

Return metadata attributes from a single spectrum or a collection of spectra as a `data.frame`. A wrapper on `add_attr2tb` providing an alternative order of formal parameters and constrained functionality.

Usage

```
spct_metadata(
  x,
  col.names = NULL,
  idx = "spct.idx",
  na.rm = is.null(col.names),
  unnest = TRUE
)
```

Arguments

`x`	generic_mspct or generic_spct Any collection of spectra or spectrum.
`col.names`	named character vector Name(s) of column(s) to create.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`na.rm`	logical Flag controlling deletion of columns containing only NA values.
`unnest`	logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.

Details

Each attribute is by default copied to a column in a `tibble` or a `data.frame`. If the argument for `tb` is `NULL`, as by default, a new `tibble` will be created. If an existing `data.frame` or `tibble` is passed as argument, new columns are added to it. However, the number of rows in the argument passed to `tb` must match the number of spectra in the argument passed to `mspct`. Only in the case of methods `add_attr2tb()` and `spct_metadata()` if the argument to `col.names` is a named vector, the names of members are used as names for the columns created. This permits setting any valid name for the new columns. If the members of the vector passed to `col.names` have no names, then the value is interpreted as the name of the attributes to add, and also used as name for the new column.

Valid values accepted as argument to `col.names` are `NULL`, or a vector containing one or more of the following `character` strings: `"lon"`, `"lat"`, `"address"`, `"geocode"`, `"where.measured"`, `"when.measured"`, `"what.measured"`, `"how.measured"`, `"comment"`, `"normalised"`, `"normalized"`, `"scaled"`, `"bswf.used"`, `"instr.desc"`, `"instr.settings"`, `solute.properties`, `"filter.properties"`, `"Tfr.type"`, `"Rfr.type"`, `"time.unit"`.

Value

A data.frame or a tibble With the metadata attributes in separate new variables.

Note

The order of the first two arguments is reversed in add_attr2tb(), when_measured2tb(), what_measured2tb(), etc., compared to attribute query functions, such as spct_metadata, when_measured(), what_measured(), how_measured(), etc. This is to allow the use of add_attr2tb() in 'pipes' to add metadata to summaries computed at earlier steps in the pipe.

See Also

add_attr2tb for more details.

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), subset_attributes(), trimInstrDesc(), trimInstrSettings()

Examples

```
# collection of spectra
spct_metadata(sun_evening.mspct)

spct_metadata(sun_evening.mspct, na.rm = FALSE)

spct_metadata(sun_evening.mspct,
              col.names = "geocode",
              unnest = FALSE)

spct_metadata(sun_evening.mspct,
              col.names = c(when.measured = "time", "what.measured"))

# multiple spectra in long form
spct_metadata(sun_evening.spct,
              col.names = c("geocode", "when.measured"))

# single spectrum
spct_metadata(sun.spct,
              col.names = c("geocode", "when.measured"))
```

---

spct_wide2long *Convert spectrum from wide to long form*

Description

Convert spectrum from wide to long form

Usage

```
spct_wide2long(
  spct,
  fixed.cols = "w.length",
  idfactor = "spct.idx",
  rm.spct.class = FALSE,
  ...
)
```

Arguments

`spct`	An object with spectral data.
`fixed.cols`	character Names of variables that should be copied unchanged for each spectrum.
`idfactor`	character The name of the factor to be added to the long-form object and used to store the original name of the columns as an index to the different spectra.
`rm.spct.class`	logical If true the returned object is a data frame.
`...`	Currently ignored.

Details

Only objects of classes raw_spct, cps_spct, and object_spct normally contain multiple columns of spectral data. These are supported as well as generic_spct. Is the wide spectra contain multiple spectra in long form, the original `idfactor` is preserved.

Spectra that are already in long form, if passed as argument, are returned unchanged.

Because the classes defined for spectra have a well defined format, and known column names we can define a rather simple function for this operation.

Value

An object of the same class as `spct` or a `data.frame` with derived classes removed.

Examples

```
spct_wide2long(white_led.raw_spct)
spct_wide2long(white_led.cps_spct)
spct_wide2long(Ler_leaf.spct)
```

spikes *Spikes*

Description

Function that returns a subset of an R object with observations corresponding to spikes. Spikes are values in spectra that are unusually high compared to neighbors. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode arrays.

Usage

```
spikes(x, z.threshold, max.spike.width, na.rm, ...)

## Default S3 method:
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'numeric'
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  y.var.name = NULL,
  var.name = y.var.name
)

## S3 method for class 'generic_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  var.name = NULL,
  ...
)

## S3 method for class 'source_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
spikes(x, z.threshold = 9, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'solute_spct'
spikes(x, z.threshold = 9, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'cps_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  var.name = "cps",
  ...
)

## S3 method for class 'raw_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  var.name = "counts",
  ...
```

```
)

## S3 method for class 'generic_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  var.name = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'source_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
```

```
)

## S3 method for class 'reflector_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  var.name = "cps",
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  var.name = "counts",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object
`z.threshold`	numeric Modified Z values larger than `z.threshold` are considered to correspond to spikes.
`max.spike.width`	integer Wider regions with high Z values are not detected as spikes.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for spikes.
`...`	ignored
`var.name`, `y.var.name`	character Name of column where to look for spikes.
`unit.out`	character One of "energy" or "photon"
`filter.qty`	character One of "transmittance" or "absorbance"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, resulting in most cases in more spikes being detected. A modified version of the algorithm is used if a value different from `NULL` is passed as argument to `max.spike.width`. In such a case, an additional step filters out broader spikes (or falsely detected steep slopes) from the returned values.

When the argument passed to `x` contains multiple spectra, the spikes are searched for in each spectrum independently of other spectra.

Value

A subset of the object passed as argument to `x` with rows corresponding to spikes.

Methods (by class)

- `spikes(default)`: Default returning always NA.
- `spikes(numeric)`: Default function usable on numeric vectors.
- `spikes(data.frame)`: Method for "data.frame" objects.
- `spikes(generic_spct)`: Method for "generic_spct" objects.
- `spikes(source_spct)`: Method for "source_spct" objects.
- `spikes(response_spct)`: Method for "response_spct" objects.
- `spikes(filter_spct)`: Method for "filter_spct" objects.
- `spikes(reflector_spct)`: Method for "reflector_spct" objects.

- `spikes(solute_spct)`: Method for "solute_spct" objects.
- `spikes(cps_spct)`: Method for "cps_spct" objects.
- `spikes(raw_spct)`: Method for "raw_spct" objects.
- `spikes(generic_mspct)`: Method for "generic_mspct" objects.
- `spikes(source_mspct)`: Method for "source_mspct" objects.
- `spikes(response_mspct)`: Method for "cps_mspct" objects.
- `spikes(filter_mspct)`: Method for "filter_mspct" objects.
- `spikes(reflector_mspct)`: Method for "reflector_mspct" objects.
- `spikes(solute_mspct)`: Method for "solute_mspct" objects.
- `spikes(cps_mspct)`: Method for "cps_mspct" objects.
- `spikes(raw_mspct)`: Method for "raw_mspct" objects.

See Also

See the documentation for `find_spikes` for details of the algorithm and implementation.

Other peaks and valleys functions: `find_peaks()`, `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `valleys()`, `wls_at_target()`

Examples

```
spikes(sun.spct)
```

`split2mspct` *Convert a 'wide' or untidy data frame into a collection of spectra*

Description

Convert a data frame object into a "multi spectrum" object by constructing a an object of a multi-spct class, converting numeric columns other than wavelength into individual spct objects.

Usage

```
split2mspct(
  x,
  member.class = NULL,
  spct.data.var = NULL,
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
```

```
split2source_mspct(
  x,
  spct.data.var = "s.e.irrad",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2response_mspct(
  x,
  spct.data.var = "s.e.response",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2filter_mspct(
  x,
  spct.data.var = "Tfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2reflector_mspct(
  x,
  spct.data.var = "Rfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2solute_mspct(
  x,
  spct.data.var = "K.mole",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2cps_mspct(
  x,
  spct.data.var = "cps",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2raw_mspct(
  x,
  spct.data.var = "count",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2calibration_mspct(
  x,
  spct.data.var = "irrad.mult",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
```

Arguments

`x`	data frame
`member.class`	character Class of the collection members
`spct.data.var`	character Name of the spectral data argument in the object constructor for `member.class`
`w.length.var`	character Name of column containing wavelength data in nanometres
`idx.var`	character Name of column containing data to be copied unchanged to each spct object
`ncol`	integer Number of 'virtual' columns in data
`byrow`	logical If `ncol` > 1 how to read in the data
`...`	additional named arguments passed to the member constructor function.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, as.reflector_mspct(), as.response_mspct(), as.solute_mspct(), as.source_mspct(), subset2mspct()

split_bands *List-of-wavebands constructor*

Description

Build a list of unweighted "waveband" objects that can be used as input when calculating irradiances.

Usage

```
split_bands(
  x,
  list.names = NULL,
  short.names = is.null(list.names),
  length.out = NULL
)
```

Arguments

x	a numeric vector of wavelengths to split at (nm), or a range of wavelengths or a generic_spct or a waveband.
list.names	character vector with names for the component wavebands in the returned list (in order of increasing wavelength)
short.names	logical indicating whether to use short or long names for wavebands
length.out	numeric giving the number of regions to split the range into (ignored if w.length is not numeric).

Value

an un-named list of waveband objects

Note

`list.names` is used to assign names to the elements of the list, while the waveband objects themselves always retain their `wb.label` and `wb.name` as generated during their creation.

See Also

Other waveband constructors: waveband()

Examples

```
split_bands(c(400,500,600))
split_bands(list(c(400,500),c(550,650)))
split_bands(list(A=c(400,500),B=c(550,650)))
split_bands(c(400,500,600), short.names=FALSE)
split_bands(c(400,500,600), list.names=c("a","b"))
split_bands(c(400,700), length.out=6)
split_bands(400:700, length.out=3)
split_bands(sun.spct, length.out=10)
split_bands(waveband(c(400,700)), length.out=5)
```

`split_energy_irradiance` *Energy irradiance for split spectrum regions*

Description

This function returns the energy irradiance for a series of contiguous wavebands from a radiation-source spectrum. The returned values can be either absolute or relative to their sum.

Usage

```
split_energy_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`s.irrad`	numeric vector of spectral (energy or photon) irradiance values (W m-2 nm-1) or (mol s-1 m-2 nm-1).
`cut.w.length`	numeric vector of wavelengths (nm).
`unit.in`	character string with allowed values "energy", and "photon", or its alias "quantum".
`scale`	character string indicating the scale used for the returned values ("absolute", "relative", "percent").
`check.spectrum`	logical indicating whether to sanity check input data, default is TRUE.

`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a numeric vector of irradiances with no change in scale factor: [W m-2 nm-1] -> [W m-2] or [mol s-1 m-2] -> [W m-2] or relative values (fraction of one) if scale = "relative" or scale = "percent".

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data,
     split_energy_irradiance(w.length, s.e.irrad,
                             cut.w.length = c(300, 400, 500, 600, 700)))
```

`split_irradiance` *Energy or photon irradiance for split spectrum regions*

Description

This function returns the energy or photon irradiance for a series of contiguous wavebands from a radiation spectrum. The returned values can be either absolute or relative to their sum.

Usage

```
split_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.out = getOption("photobiology.base.unit", default = "energy"),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric Vector of wavelengths $[nm]$.
`s.irrad`	numeric vector of spectral irradiances in $[W\,m^{-2}\,nm^{-1}]$ or $[mol\,s^{-1}\,sm^{-2}\,nm^{-1}]$ as indicated by the argument pased to `unit.in`.
`cut.w.length`	numeric Vector of wavelengths $[nm]$.
`unit.out`, `unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`scale`	a character A string indicating the scale used for the returned values (`"absolute"`, `"relative"` or `"percent"`).
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is `TRUE`.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A numeric vector of irradiances with no change in scale factor if `scale == "absolute"`, $[W\,m^{-2}]$ or $[mol\,s^{-1}\,sm^{-2}]$ depending on the argument passed to `unit.out` or relative values (as fraction of one if `scale == "relative"` or percentages if `scale == "percent"` of photons or energy depending on the argument passed to `unit.out`.

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

Examples

```
with(sun.data,
     split_irradiance(w.length, s.e.irrad,
                      cut.w.length = c(300, 400, 500, 600, 700),
                      unit.out = "photon"))
```

split_photon_irradiance

Photon irradiance for split spectrum regions

Description

This function returns the photon irradiance for a series of contiguous wavebands from a radiation spectrum. The returned values can be either absolute or relative to their sum.

Usage

```
split_photon_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`s.irrad`	numeric vector of spectral (energy or photon) irradiance values (W m-2 nm-1).
`cut.w.length`	numeric vector of wavelengths (nm).
`unit.in`	character Allowed values "energy", and "photon", or its alias "quantum".
`scale`	a character A string indicating the scale used for the returned values ("absolute", "relative", "percent").
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is TRUE.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a numeric vector of photon irradiances with no change in scale factor: [W m-2 nm-1] -> [mol s-1 m-2], [mol s-1 m-2 nm-1] -> [mol s-1 m-2] or relative values (fraction of one based on photon units) if scale = "relative" or scale = "percent".

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
with(sun.data,
     split_photon_irradiance(w.length, s.e.irrad,
             cut.w.length = c(300, 400, 500, 600, 700)))
with(sun.data,
     split_photon_irradiance(w.length, s.e.irrad))
```

`spread` *Expanse*

Description

A method that returns the expanse ($max(x) - min(x)$) for R objects. In particular the wavelength $[nm]$ expanse of the wavelength range of objects of classes `waveband` or of class `generic_spct` or derived (or the expanse of values in a `numeric` vector).

Usage

```
spread(x, ...)

wl_expanse(x, ...)

expanse(x, ...)
```

```
## Default S3 method:
expanse(x, ...)

## S3 method for class 'numeric'
expanse(x, ...)

## S3 method for class 'waveband'
expanse(x, ...)

## S3 method for class 'generic_spct'
expanse(x, ...)

## S3 method for class 'generic_mspct'
expanse(x, ..., idx = "spct.idx")
```

Arguments

`x`	an R object
`...`	not used in current version
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

A numeric value equal to `max(x) - min(x)`. In the case of spectral objects wavelength difference $[nm]$. For any other R object, according to available specialised methods of `min` and `max`.

Methods (by class)

- `expanse(default)`: Default method for generic function
- `expanse(numeric)`: Method for "numeric"
- `expanse(waveband)`: Method for "waveband"
- `expanse(generic_spct)`: Method for "generic_spct"
- `expanse(generic_mspct)`: Method for "generic_mspct" objects.

Examples

```
expanse(10:20)
expanse(sun.spct)
wl_expanse(sun.spct)

expanse(sun.spct)
```

Subset *Subsetting spectra*

Description

Return subsets of spectra stored in class `generic_spct` or derived from it.

Usage

```
## S3 method for class 'generic_spct'
subset(x, subset, select, drop = FALSE, ...)
```

Arguments

`x`	object to be subsetted.
`subset`	logical expression indicating elements or rows to keep: missing values are taken as false.
`select`	expression, indicating columns to select from a spectrum.
`drop`	passed on to `[` indexing operator.
`...`	further arguments to be passed to or from other methods.

Value

An object similar to `x` containing just the selected rows and columns. Depending on the columns remaining after subsetting the class of the object will be simplified to the most derived parent class.

Note

This method is copied from `base::subset.data.frame()` but ensures that all metadata stored in attributes of spectral objects are copied to the returned value.

Examples

```
subset(sun.spct, w.length > 400)
```

subset2mspct *Convert 'long' or tidy spectral data into a collection of spectra*

Description

Convert a data frame object or spectral object into a collection of spectra object of the matching class. For data frames converting numeric columns other than wavelength into individual spct objects. For collection of spectra objects, subset/expand long-form members into multiple members of the same collection.

Usage

```
subset2mspct(
  x,
  member.class = NULL,
  idx.var = NULL,
  drop.idx = TRUE,
  ncol = 1,
  byrow = FALSE,
  ...
)
```

Arguments

`x`	a generic_spct object or of a derived class, or a data frame, or a generic_mspct object or of a derived class.
`member.class`	character string.
`idx.var`	character Name of column containing data to be copied unchanged to each spct object or used for member names. If `NULL`, the default, the name is retrieved from x or its members when possible.
`drop.idx`	logical Flag indicating whether to drop or keep idx.var in the collection members.
`ncol`	integer Number of 'virtual' columns in data.
`byrow`	logical If `ncol` > 1 how to read in the data.
`...`	additional named arguments passed to the member constructor function.

Value

A collection of spectral objects, each with attributes set if x is a spectral object in long form with metadata attributes. If this object was created by row binding with 'photobiology' 0.9.14 or later then all metadata for each individual spectrum will be preserved, except for unique comments which are merged.

Note

A non-null value for `member.class` is mandatory only when `x` is a data frame.

See Also

Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.reflector_mspct()`, `as.response_mspct()`, `as.solute_mspct()`, `as.source_mspct()`, `split2mspct()`

`subt_spectra` *Subtract two spectra*

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added. This is 'parallel' operation between two spectra.

Usage

```
subt_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

Arguments

`w.length1`	numeric vector of wavelength (nm).
`w.length2`	numeric vector of wavelength (nm).
`s.irrad1`	a numeric vector of spectral values.
`s.irrad2`	a numeric vector of spectral values.
`trim`	a character string with value "union" or "intersection".
`na.rm`	a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a data frame with two numeric variables

`w.length`	A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique vales, sorted in ascending order.
`s.irrad`	A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
head(sun.data)
zero.data <- with(sun.data, subt_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(zero.data)
tail(zero.data)
```

`summary.generic_spct` *Summary of one or more spectra*

Description

Methods of generic function summary for objects of spectral classes and of classes for collections of spectra.

Usage

```
## S3 method for class 'generic_spct'
summary(
  object,
  maxsum = 7,
  digits = max(3, getOption("digits") - 3),
  ...,
  expand = "none"
)

## S3 method for class 'generic_mspct'
summary(
  object,
  maxsum = 7,
  digits = max(3, getOption("digits") - 3),
  idx = "spct.idx",
  which.metadata = NULL,
  expand = "none",
  ...
)
```

Arguments

`object`	An object of one of the spectral classes for which a summary is desired.
`maxsum`	integer Indicates how many levels should be shown for factors.
`digits`	integer Used for number formatting with `format()`.
`...`	additional arguments affecting the summary produced, ignored in current version.
`expand`	character One of `"none"`, `"collection"`, `"each"` or `"auto"` indicating if multiple spectra in long form should be summarized as a collection or individually.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`which.metadata`	character vector Names of attributes to retrieve, or "none" or "all". Obeyed if `expand = "collection"`, its default.

Details

Objects are summarized as is, ignoring the current settings of R options `photobiology.radiation.unit` and `photobiology.filter.qty`. Unlike R's summary, these methods can optionally summarize each spectrum stored in long form returning a list of summaries. Although this is frequently the most informative approach, the default remains similar to `summary()` method from R: to summarize `object` as a whole. Alternatively, multiple spectra stored in long form, can optionally be summarized also as a collection of spectra. Passing `"auto"` in the call, is equivalent to passing `"each"` or `"collection"` depending on the number of spectra contained in the object.

Value

A summary object matching the class of `object`, or a list of such objects or a summary object for a matching collection of spectra. Metadata stored in attributes are copied to identical attributes in the returned summary objects except when `object` is a collection of spectra or if `expand = "collection"` is passed in the call. In this two cases, a condensed summary is returned as a data frame and attributes from each member can be copied to variables in it.

Functions

- `summary(generic_mspct)`:

See Also

`print.summary_generic_spct`

Examples

```
summary(sun.spct)
class(summary(sun.spct))

summary(two_filters.spct)
class(summary(two_filters.spct))

summary(sun_evening.spct)
summary(two_filters.spct, expand = "none")
summary(two_filters.spct, expand = "each")
summary(two_filters.spct, expand = "collection")
summary(two_filters.spct, expand = "auto") # <= 4 spectra
summary(sun_evening.spct, expand = "auto") # > 4 spectra

where_measured(sun.spct)
where_measured(summary(sun.spct))
what_measured(summary(two_filters.spct))
what_measured(summary(two_filters.spct, expand = "each")[[1]])

summary(sun_evening.mspct)
summary(sun_evening.mspct, which.metadata = "when.measured")
summary(two_filters.mspct, which.metadata = "what.measured")
summary(two_filters.mspct, expand = "each")
```

`summary_spct_classes` *Function that returns a vector containing the names of spectral summary classes.*

Description

Function that returns a vector containing the names of spectral summary classes.

Usage

```
summary_spct_classes()
```

Value

A `character` vector of class names.

`sum_spectra` *Add two spectra*

Description

Merge wavelength vectors of two spectra, and compute the missing spectral values by interpolation within each spectrum. After this, the spectral values at each wavelength are added. This is a 'parallel' operation between two spectra.

Usage

```
sum_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

Arguments

`w.length1`	numeric vector of wavelength (nm).
`w.length2`	numeric vector of wavelength (nm).
`s.irrad1`	a numeric vector of spectral values.
`s.irrad2`	a numeric vector of spectral values.
`trim`	a character string with value "union" or "intersection".
`na.rm`	a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If `w.length2` = `NULL`, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a `data.frame` with two numeric variables

`w.length`	A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique vales, sorted in ascending order.
`s.irrad`	A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
head(sun.data)
twice.sun.data <- with(sun.data, sum_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(twice.sun.data)
tail(twice.sun.data)
```

`sun.spct` *Solar spectral irradiance (simulated)*

Description

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance and spectral photon irradiance. Values simulated for 22 June 2010, near midday, at Helsinki, under partly cloudy conditions. The variables are as follows:

Usage

```
sun.spct

sun.data
```

Format

A `source_spct` object and a `data.frame`, each with 511 rows and 3 variables

An object of class `data.frame` with 508 rows and 3 columns.

Details

- w.length (nm), range 293 to 800 nm.
- s.e.irrad (W m-2 nm-1)
- s.q.irrad (mol m-2 nm-1)

Note

Package ’photobiologySun’ contains data sets for the daylight spectrum under different conditions in and outside vegetation, stored in objects of these same classes, ready to be used with package ’photobiology’.

Author(s)

Anders K. Lindfors (data)

References

Lindfors, A.; Heikkilä, A.; Kaurola, J.; Koskela, T. & Lakkala, K. (2009) Reconstruction of Solar Spectral Surface UV Irradiances Using Radiative Transfer Simulations. Photochemistry and Photobiology, 85: 1233-1239

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
sun.spct
summary(sun.spct)
```

`sun_daily.spct` *Daily solar spectral irradiance (simulated)*

Description

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance. Values simulated for 2 June 2012, at Helsinki, under clear sky conditions. The variables are as follows:

Usage

```
sun_daily.spct

sun_daily.data

sun.daily.spct

sun.daily.data
```

Format

A `source_spct` object and a `data.frame`, each with 511 rows and 3 variables

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 511 rows and 3 columns.

An object of class `source_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 522 rows and 3 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 511 rows and 3 columns.

Details

- w.length (nm), range 290 to 800 nm.
- s.e.irrad (J d-1 m-2 nm-1)
- s.q.irrad (mol d-1 m-2 nm-1)

Deprecation!

Objects `sun.daily.spct` and `sun.daily.data` have been renamed into `sun_daily.spct` and `sun_daily.data`, for consistency with other data sets in the package. Please, use the new names for new code.

Note

The simulations are based on libRadTran using hourly mean global radiation measurements to estimate cloud cover. The simulations were for each hour and the results integrated for the whole day.

Author(s)

Anders K. Lindfors (data)

References

Lindfors, A.; Heikkilä, A.; Kaurola, J.; Koskela, T. & Lakkala, K. (2009) Reconstruction of Solar Spectral Surface UV Irradiances Using Radiative Transfer Simulations. Photochemistry and Photobiology, 85: 1233-1239

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
sun.daily.spct
summary(sun.daily.spct)
```

`sun_evening.spct` *Time series of solar spectral irradiance (measured)*

Description

Two data objects containing containing the same time series of five spectra. Values measured in Viikki, Helsinki, under nearly clear sky in a summer evening.

Usage

```
sun_evening.spct

sun_evening.mspct
```

Format

A `source_spct` object and a `source_mspct` object.

An object of class `source_mspct` (inherits from `generic_mspct`, `list`) with 5 rows and 1 columns.

Details

The variables are as follows:

- w.length (nm), range 290 to 1000 nm.
- s.e.irrad (J d-1 m-2 nm-1)
- s.q.irrad (mol d-1 m-2 nm-1)

Author(s)

Pedro J. Aphalo (data)

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
summary(sun_evening.mspct)
colnames(sun_evening.spct)
```

`s_e_irrad2rgb`	*Spectral irradiance to rgb color conversion*

Description

Calculates rgb values from spectra based on human color matching functions (CMF) or chromaticity coordinates (CC). A CMF takes into account luminous sensitivity, while a CC only the color hue. This function, in contrast to that in package pavo does not normalize the values to equal luminosity, so using a CMF as input gives the expected result. Another difference is that it allows the user to choose the chromaticity data to be used. The data used by default is different, and it corresponds to the whole range of CIE standard, rather than the reduced range 400 nm to 700 nm. The wavelength limits are not hard coded, so the function could be used to simulate vision in other organisms as long as pseudo CMF or CC data are available for the simulation.

Usage

```
s_e_irrad2rgb(
  w.length,
  s.e.irrad,
  sens = photobiology::ciexyzCMF2.spct,
  color.name = NULL,
  check = TRUE
)
```

Arguments

`w.length`	numeric vector of wavelengths (nm).
`s.e.irrad`	numeric vector of spectral irradiance values.
`sens`	a chroma_spct object with variables w.length, x, y, and z, giving the CC or CMF definition (default is the proposed human CMF according to CIE 2006.).
`color.name`	character string for naming the rgb color definition.
`check`	logical indicating whether to check or not spectral data.

Value

A color defined using `rgb`. The numeric values of the RGB components can be obtained using function `col2rgb`.

Note

Very heavily modified from Chad Eliason's <`cme16@zips.uakron.edu`> spec2rgb function in package `Pavo`.

References

CIE(1932). Commission Internationale de l'Eclairage Proceedings, 1931. Cambridge: Cambridge University Press.

Color matching functions obtained from Colour and Vision Research Laboratory online data repository at http://www.cvrl.org/.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
my.color <-
    with(sun.data,
         s_e_irrad2rgb(w.length, s.e.irrad, color.name = "sunWhite"))
col2rgb(my.color)
```

---

s_mean *Mean from collection of spectra*

Description

Method to compute the "parallel" mean of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_mean(x, trim, na.rm, ...)

## Default S3 method:
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

```
## S3 method for class 'filter_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`trim`	numeric The fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the mean at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiolgy'. Trimming of extreme values is possible (trimmed mean) and omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `mean` for the `mean()` method used for the computations.

Examples

```
s_mean(sun_evening.mspct)
```

---

s_mean_se *Mean and standard error from collection of spectra*

Description

Method to compute the "parallel" mean and the SEM. The spectral values are summarised across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_mean_se(x, na.rm, mult, ...)

## Default S3 method:
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'generic_spct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'filter_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'source_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'response_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'reflector_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'calibration_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)
```

```
## S3 method for class 'cps_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'raw_mspct'
s_mean_se(x, na.rm = FALSE, mult = 1, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`mult`	numeric number of multiples of standard error.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the mean and SEM at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled. The value passed as argument to 'mult'

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `mean` for the `mean()` method to compute the mean and `sd` for the method used to compute the standard error of the mean.

Examples

```
s_mean_se(sun_evening.mspct)
```

`s_mean_se_band` *Mean plus and minus standard error from collection of spectra*

Description

Method to compute the "parallel" mean and limits based on SEM. The spectral values are summarised across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_mean_se_band(x, na.rm, mult, ...)

## Default S3 method:
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'generic_spct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'filter_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'source_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'response_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'reflector_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'calibration_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'cps_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'raw_mspct'
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`mult`	numeric number of multiples of standard error.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the mean and limits based on SEM at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled. The value passed as argument to 'mult' can be used to estimate a confidence interval for each mean value.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `mean` for the `mean()` method used for the computations.

Examples

```
s_mean_se_band(sun_evening.mspct)
```

`s_median` *Median of a collection of spectra*

Description

Method to compute the "parallel" median of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_median(x, na.rm, ...)

## Default S3 method:
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_median(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the median at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `median` for the `median()` method used for the computations.

Examples

```
s_median(sun_evening.mspct)
```

`s_prod` *Product from collection of spectra*

Description

Method to compute the "parallel" product of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_prod(x, na.rm, ...)

## Default S3 method:
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_prod(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the product at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Note

The product operation is meaningful only for certain physical quantities or bases of expression.

See Also

See `prod` for the `prod()` method used for the computations.

Examples

```
s_prod(two_filters.mspct)
```

---

s_quantile *Quantiles of a collection of spectra*

Description

Method to compute the "parallel" quantiles of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_quantile(x, probs, na.rm, ...)

## Default S3 method:
s_quantile(x, probs = NA, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'source_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'response_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'filter_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'reflector_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'calibration_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)
```

```
## S3 method for class 'cps_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)

## S3 method for class 'raw_mspct'
s_quantile(x, probs = c(0.25, 0.5, 0.75), na.rm = FALSE, ..., simplify = TRUE)
```

Arguments

`x`	An R object.
`probs`	numeric vector of probabilities with values in $[0, 1]$.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.
`simplify`	logical If `TRUE` and a single quantile is computed, return an spectrum by itself instead of as a single member of a collection.

Details

Method specializations compute the qunatiles at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is not fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object is of same class as the the collection, such as `"filter_mspct"`, containing one member summary spectrum for each value in `probs` with the same variable names as in the input. If a single quantile is computed and `simplify` = `TRUE` a single spectrum such as `"filter_spct"` is returned.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `quantile` for the `quantile` method used for the computations. Additional arguments can be passed by name to be forwarded to `quantile`.

Examples

```
s_quantile(sun_evening.mspct)
```

`s_range` *Range of a collection of spectra*

Description

Method to compute the "parallel" range of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_range(x, na.rm, ...)

## Default S3 method:
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_range(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_range(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.

`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the range at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Deepest Curves

Parallel summaries differ fundamentally from the "deepest curves" obtained through functional data analysis (FDA) in that in functional data analysis one of the input curves is returned as the deepest one based on a decision criterion. In contrast the parallel summaries from package 'photobioloy' return one or more "fictional" curves different to any of those passed as inputs. This curve is constructed from independent summaries at each wavelength value.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `Extremes` for details on the `min()` and `max()` methods used for the computations.

Examples

```
s_range(sun_evening.mspct)
```

s_sd *Standard Deviation of a collection of spectra*

Description

Method to compute the "parallel" standard deviation of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_sd(x, na.rm, ...)

## Default S3 method:
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_sd(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the standard deviation at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `sd` for details about `sd()` methods for other classes.

Examples

```
s_sd(sun_evening.mspct)
```

---

s_se *Standard Error of a collection of spectra*

Description

Method to compute the "parallel" standard error of the mean across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_se(x, na.rm, ...)

## Default S3 method:
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
```

```
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_se(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the standard error of the mean at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See sd for details about sd() methods for other classes.

Examples

```
s_se(sun_evening.mspct)
```

s_sum *Sum from collection of spectra*

Description

Method to compute the "parallel" sum of values across members of a collection of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_sum(x, na.rm, ...)

## Default S3 method:
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_sum(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_sum(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the sum at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Note

The sum operation is meaningful only for certain physical quantities or bases of expression.

See Also

See `sum` for the `sum()` method used for the computations.

Examples

```
s_sum(sun_evening.mspct)
```

`s_var` *Variance of a collection of spectra*

Description

Method to compute the "parallel" variance of values across members of a collections of spectra or of a spectral object containing multiple spectra in long form.

Usage

```
s_var(x, na.rm, ...)

## Default S3 method:
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'generic_spct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_var(x, na.rm = FALSE, ...)
```

Arguments

`x`	An R object.
`na.rm`	logical A value indicating whether NA values should be stripped before the computation proceeds.
`...`	Further arguments passed to or from other methods.

Details

Method specializations compute the variance at each wavelength across a group of spectra stored in an object of one of the classes defined in package 'photobiology'. Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths. An error is triggered if this condition is nor fulfilled.

Value

If `x` is a collection spectral of objects, such as a `"filter_mspct"` object, the returned object belongs to the same class as the members of the collection, such as `"filter_spct"`, containing the summary spectrum, with variables with names tagged for summaries other than mean or median.

Note

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans in multiple variables or "columns". The parallel summaries' methods accept as arguments objects of these classes only if spectra contain data for a single spectrometer scan. In the case of `cps_spct` objects, a single column can also contain data from multiple scans spliced into a single variable.

See Also

See `cor` for details about `var()`, which is used for the computations.

Examples

```
s_var(sun_evening.mspct)
```

`T2A` *Convert transmittance into absorbance.*

Description

Function that converts transmittance (fraction) into $\log_{10}$-based absorbance (a.u.).

Usage

```
T2A(x, action, byref, clean, ...)

## Default S3 method:
T2A(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'numeric'
T2A(x, action = NULL, byref = FALSE, clean = TRUE, ...)

## S3 method for class 'filter_spct'
T2A(x, action = "add", byref = FALSE, clean = TRUE, strict.A = FALSE, ...)

## S3 method for class 'filter_mspct'
T2A(
  x,
  action = "add",
  byref = FALSE,
  clean = TRUE,
  strict.A = TRUE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object.
`action`	character Allowed values `"replace"` and `"add"`.
`byref`	logical indicating if new object will be created by reference or by copy of `x`.
`clean`	logical replace off-boundary values before conversion
`...`	not used in current version
`strict.A`	logical Attempt to compute a true internal absorbance even if `"total"` transmittance is stored in `x`.
`.parallel`	if `TRUE`, apply function in parallel, using parallel backend provided by foreach.
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Absorbance, A, is frequently used in chemistry as it is linearly related to the concentration of a solute dissolved in a solvent.

$$A = -\log_{10} \tau$$

where, A absorbance and τ is internal transmittance. By default, if total transmittance, T, is stored in `x`, the returned value computed as

$$A = -\log_{10} T$$

is not strictly absorbance. In this case and in cases when the measured light attenuation is the result of scattering, or when part of measured light is re-emitted after absorption the use of *attenuance* is the IUPAC-recommended name for this quantity.

If `strict.A = TRUE` is passed in the call and total transmittance, T, and total reflectance, ρ, are both available, absorbance is computed as:

$$A = -\log_{10}(T - \rho)/(1 - \rho)$$

where ρ can be either spectral total reflectance stored in `x` as data or a single approximate `Rfr.constant` value stored as part of the metadata.

Value

A copy of `x` with a column `A` added and other columns possibly deleted except for `w.length`. If `action = "replace"`, in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- `T2A(default)`: Default method for generic function
- `T2A(numeric)`: Method for numeric vectors
- `T2A(filter_spct)`: Method for filter spectra
- `T2A(filter_mspct)`: Method for collections of filter spectra

Note

The default `A.strict = FALSE` ensures indentical behaviour as in 'photobiology' (<= 0.11.0).

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

`T2Afr` *Convert transmittance into absorptance.*

Description

Function that converts transmittance (fraction) into absorptance (fraction). If reflectance (fraction) is available, it also allows conversions between internal and total absorptance.

Usage

```
T2Afr(x, action, byref, clean, ...)

## Default S3 method:
T2Afr(x, action = NULL, byref = FALSE, clean = FALSE, ...)

## S3 method for class 'numeric'
T2Afr(x, action = NULL, byref = FALSE, clean = FALSE, Rfr = NA_real_, ...)

## S3 method for class 'filter_spct'
T2Afr(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'object_spct'
T2Afr(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'filter_mspct'
T2Afr(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
T2Afr(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object.
`action`	character Allowed values `"replace"` and `"add"`.
`byref`	logical indicating if new object will be created by reference or by copy of `x`.
`clean`	logical replace off-boundary values before conversion.
`...`	not used in current version.
`Rfr`	numeric vector. Spectral reflectance o reflectance factor. Set to zero if `x` is internal reflectance,
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

Absorptance, internal transmittance and total reflectance when expressed as fractions, add up to one:

$$1 = \alpha + \tau + \rho$$

where, α is absorptance, τ is internal transmittance and ρ is total reflectance. If any two of these quantities are known, the third one can be computed from them.

On the other hand:

$$1 = \alpha\prime + T$$

where, $\alpha\prime = \alpha + \rho$, measured together. In this case, there is not enough information available to compute α.

Thus, method `T2Afr()` computes either α or $\alpha\prime$, depending on whether τ or T are contained in the argument passed to `x`, but neither of them when only τ is known. To know which quantity has been computed, use `getTfrType()` to query whether the computations were based on τ or T.

The R names used are: `Tfr` for τ and T are `Tfr`, `Afr` for α and $\alpha\prime$, and `Rfr` for rho. The distinction between τ and T and between α and $\alpha\prime$ is made based on metadata attributes.

Value

A copy of `x` with a column `Afr` added and other columns possibly deleted except for `w.length`. If `action = "replace"`, in all cases, the redundant columns are removed, even when column `Afr` was present in the argument passed to `x`.

Methods (by class)

- `T2Afr(default)`: Default method for generic function
- `T2Afr(numeric)`: Default method for generic function
- `T2Afr(filter_spct)`: Method for filter spectra
- `T2Afr(object_spct)`: Method for object spectra
- `T2Afr(filter_mspct)`: Method for collections of filter spectra
- `T2Afr(object_mspct)`: Method for collections of object spectra

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `any2T()`, `as_quantum()`, `e2q()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

Examples

```
T2Afr(Ler_leaf.spct)
```

`tag` *Tag a spectrum*

Description

Spectra are tagged by adding variables and attributes containing color definitions, labels, and a factor following the wavebands given in `w.band`. This methods are most useful for plotting realistic computed colors from spectral data.

Usage

```
tag(x, ...)

## Default S3 method:
tag(x, ...)

## S3 method for class 'generic_spct'
tag(
  x,
  w.band = NULL,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = TRUE,
  short.names = TRUE,
  chroma.type = "CMF",
  byref = FALSE,
  ...
)

## S3 method for class 'generic_mspct'
tag(
  x,
  w.band = NULL,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = TRUE,
  short.names = TRUE,
  chroma.type = "CMF",
  byref = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	an R object.
`...`	ignored (possibly used by derived methods).
`w.band`	waveband or list of waveband objects. The waveband(s) determine the region(s) of the spectrum that are tagged
`wb.trim`	logical Flag telling if wavebands crossing spectral data boundaries are trimmed or ignored
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`short.names`	logical Flag indicating whether to use short or long names for wavebands
`chroma.type`	character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class `chroma_spct` for any other trichromic visual system.

`byref`	logical Flag indicating if new object will be created *by reference* or *by copy* of `x`
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of `x` expanded with additional columns with color-related information.

Methods (by class)

- `tag(default)`: Default method for generic
- `tag(generic_spct)`: Tag one of `generic_spct`, and derived classes including `source_spct`, `filter_spct`, `reflector_spct`, `object_spct`, and `response_spct`.
- `tag(generic_mspct)`: Tag one of `generic_mspct`, and derived classes including `source_mspct`, `filter_mspct`, `reflector_mspct`, `object_mspct`, and `response_mspct`.

Note

`NULL` as `w.band` argument does not add any new tags, instead it removes existing tags if present. `NA`, the default, as `w.band` argument removes existing waveband tags if present and sets the `wl.color` variable. If a waveband object or a list of wavebands is supplied as argument then tagging is based on them, and `wl.color` is also set.

See Also

Other tagging and related functions: `is_tagged()`, `untag()`, `wb2rect_spct()`, `wb2spct()`, `wb2tagged_spct()`

Examples

```
tag(sun.spct)
tag(sun.spct, list(A = waveband(c(300,3005))))
```

`Tfr_fraction` *transmittance:transmittance fraction*

Description

Transmittance fraction for a given pair of wavebands of a filter spectrum.

Usage

```
Tfr_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Tfr_fraction(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'filter_spct'
Tfr_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'filter_mspct'
Tfr_fraction(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "filter_spct".
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the fraction(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the fraction(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral transmittance**, one computed for each waveband.

$$\frac{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{num}})}{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{denom}}) + \overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{num}})}$$

If the argument is set to `quantity = "total"` the fraction is based on two **integrated transmittance**, one computed for each waveband.

$$\frac{\mathrm{Tfr}(s, wb_{\mathrm{num}})}{\mathrm{Tfr}(s, wb_{\mathrm{denom}}) + \mathrm{Tfr}(s, wb_{\mathrm{num}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Tfr:Tfr]" is appended if `quantity = "total"` and "[Tfr(wl):Tfr(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Tfr_fraction(default)`: Default for generic function
- `Tfr_fraction(filter_spct)`: Method for `filter_spct` objects
- `Tfr_fraction(filter_mspct)`: Calculates Tfr:Tfr from a `filter_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other transmittance ratio functions: `Tfr_normdiff()`, `Tfr_ratio()`

Examples

```
Tfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"))
Tfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "total")
Tfr_fraction(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "mean")
```

`Tfr_normdiff` *transmittance:transmittance normalised difference*

Description

Transmittance normalized difference index for a given pair of wavebands computed from a filter spectrum.

Usage

```
Tfr_normdiff(
  spct,
  w.band.plus,
  w.band.minus,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Tfr_normdiff(
  spct,
  w.band.plus,
  w.band.minus,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)
```

```
## S3 method for class 'filter_spct'
Tfr_normdiff(
  spct,
  w.band.plus = NULL,
  w.band.minus = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'filter_mspct'
Tfr_normdiff(
  spct,
  w.band.plus = NULL,
  w.band.minus = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "filter_spct".
`w.band.plus, w.band.minus`	waveband object(s) or a list(s) of waveband objects used to compute the additive and subtractive transmittance terms of the normalized difference index.
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before

	integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral photon transmittances**, one computed for each waveband.

$$\frac{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{plus}}) - \overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{minus}})}{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{plus}}) + \overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{minus}})}$$

If the argument is set to `quantity = "total"` the fraction is based on two **photon transmittances**, one computed for each waveband.

$$\frac{\mathrm{Tfr}(s, wb_{\mathrm{plus}}) - \mathrm{Tfr}(s, wb_{\mathrm{minus}})}{\mathrm{Tfr}(s, wb_{\mathrm{plus}}) + \mathrm{Tfr}(s, wb_{\mathrm{minus}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Tfr:Tfr]" is appended if `quantity= "total"` and "[Tfr(wl):Tfr(wl)]" if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Tfr_normdiff(default)`: Default for generic function
- `Tfr_normdiff(filter_spct)`: Method for `filter_spct` objects
- `Tfr_normdiff(filter_mspct)`: Calculates Tfr:Tfr from a `filter_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult =T RUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

`normalized_diff_ind`, accepts different summary functions.

Other transmittance ratio functions: `Tfr_fraction()`, `Tfr_ratio()`

Examples

```
Tfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"))
Tfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "total")
Tfr_normdiff(Ler_leaf_rflt.spct,
             waveband(c(400,500), wb.name = "Blue"),
             waveband(c(600,700), wb.name = "Red"),
             quantity = "mean")
```

`Tfr_ratio` *transmittance:transmittance ratio*

Description

Transmittance ratio for a given pair of wavebands of a filter spectrum.

Usage

```
Tfr_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
Tfr_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'filter_spct'
Tfr_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...
)

## S3 method for class 'filter_mspct'
Tfr_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  quantity = "mean",
  naming = "short",
  name.tag = NULL,
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "filter_spct".
`w.band.num`	waveband object or a list of waveband objects used to compute the numerator(s) and denominator(s) of the ratio(s).
`w.band.denom`	waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
`scale.factor`	numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
`wb.trim`	logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
`use.cached.mult`	logical indicating whether multiplier values should be cached between calls
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	other arguments (possibly ignored)
`quantity`	character One of "total", "average" or "mean".
`naming`	character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
`name.tag`	character Used to tag the name of the returned values.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

With the default `quantity = "mean"` or `quantity = "average"` the ratio is based on two **mean spectral transmittance**, one computed for each waveband.

$$\frac{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{num}})}{\overline{\mathrm{Tfr}_\lambda}(s, wb_{\mathrm{denom}}))}$$

If the argument is set to `quantity = "total"` the ratio is based on two **integrated transmittance**, one computed for each waveband.

$$\frac{\mathrm{Tfr}(s, wb_{\mathrm{num}})}{\mathrm{Tfr}(s, wb_{\mathrm{denom}})}$$

Only if the wavelength expanse of the two wavebands is the same, these two ratios are numerically identical.

Value

In the case of methods for individual spectra, a `numeric` vector with name attribute set. The name is based on the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used. "[Tfr:Tfr]" is appended if `quantity = "mean"` or `quantity = "average"`.

A `data.frame` is returned in the case of collections of spectra, containing one column for each fraction definition, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Fraction definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`. If both arguments are lists of waveband definitions, with an equal number of members, then the wavebands are paired to obtain as many fractions as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- `Tfr_ratio(default)`: Default for generic function
- `Tfr_ratio(filter_spct)`: Method for `filter_spct` objects
- `Tfr_ratio(filter_mspct)`: Calculates Tfr:Tfr from a `filter_mspct` object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

Other transmittance ratio functions: `Tfr_fraction()`, `Tfr_normdiff()`

Examples

```
Tfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"))
Tfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"),
          quantity = "total")
Tfr_ratio(Ler_leaf_rflt.spct,
          waveband(c(400,500), wb.name = "Blue"),
          waveband(c(600,700), wb.name = "Red"),
          quantity = "mean")
```

---

thin_wl *Thin the density of wavelength values*

Description

Increase the wavelength step in stored spectral data in featureless regions to save storage space.

Usage

```
thin_wl(x, ...)

## Default S3 method:
thin_wl(x, ...)

## S3 method for class 'generic_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  col.names,
  ...
)

## S3 method for class 'source_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)
```

```
## S3 method for class 'filter_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, span = 21, ...)

## S3 method for class 'solute_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, span = 21, ...)

## S3 method for class 'raw_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  col.names,
  ...
)

## S3 method for class 'cps_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  col.names,
  ...
)

## S3 method for class 'object_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  span = 21,
  col.names,
  ...
)

## S3 method for class 'chroma_spct'
```

```
thin_wl(x, ...)

## S3 method for class 'calibration_spct'
thin_wl(x, ...)

## S3 method for class 'generic_mspct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, span = 21, ...)

## S3 method for class 'chroma_mspct'
thin_wl(x, ...)

## S3 method for class 'calibration_mspct'
thin_wl(x, ...)
```

Arguments

`x`	An R object
`...`	additional named arguments passed down to `f`.
`max.wl.step`	numeric. Largest allowed wavelength difference between adjacent spectral values in nanometres (nm).
`max.slope.delta`	numeric in 0 to 1. Largest allowed change in relative slope of the spectral quantity per nm between adjacent pairs of values.
`span`	integer A peak (or valley) is defined as an element in a sequence which is greater (or smaller) than all other elements within a window of width span centred at that element. Use NULL for the global peak.
`col.names`	character. Name of the column of `x` containing the spectral data to check against `max.slope.delta`. Currently only one column supported.
`unit.out`	character Allowed values "energy", and "photon", or its alias "quantum".
`qty.out`	character Allowed values "transmittance", and "absorbance".

Details

The algorithm used for spectra is "naive" in an effort to keep it efficient. It works by iteratively attempting to delete every other observation along wavelengths, based on the criteria for maximum wavelength step and maximum relative step in the spectral variable between adjacent data values.

Value

An object of the same class as `x` but with a reduced density of wavelength values in those regions were slope is shallow and featureless.

Methods (by class)

- `thin_wl(default)`: Default for generic function
- `thin_wl(generic_spct)`:
- `thin_wl(source_spct)`:

- `thin_wl(response_spct)`:
- `thin_wl(filter_spct)`:
- `thin_wl(reflector_spct)`:
- `thin_wl(solute_spct)`:
- `thin_wl(raw_spct)`:
- `thin_wl(cps_spct)`:
- `thin_wl(object_spct)`:
- `thin_wl(chroma_spct)`:
- `thin_wl(calibration_spct)`:
- `thin_wl(generic_mspct)`:
- `thin_wl(chroma_mspct)`:
- `thin_wl(calibration_mspct)`:

Note

The value of `max.slope.delta` is expressed as relative change in the slope of spectral variable per nanometre. This means that values between 0.0005 and 0.005 tend to work reasonably well. The best value will depend on the wavelength step of the input and noise in data. A moderate smoothing before thinning can sometimes help in the case of noisy data.

The amount of thinning is almost always less than the value of criteria passed as argument as it is based on existing wavelength values. For example if we start with a spectrum with a uniform wavelength step of 1 nm, possible steps in the thinned spectrum are 2, 4, 8, 16, 32, etc. nm. The algorithm, does work with any step sizes, regular or variable in the input. Thinning is most effective for spectra with large "featureless" regions as the algorithm attempts not to discard information, contrary to smoothing or interpolation.

Local peaks and valleys are always preserved, using by default a span of 21 to search for them. See `find_peaks`.

See Also

Other experimental utility functions: `collect2mspct()`, `drop_user_cols()`, `uncollect2spct()`

Examples

```
nrow(yellow_gel.spct)
wl_stepsize(yellow_gel.spct)
thinned.spct <- thin_wl(yellow_gel.spct)
nrow(thinned.spct)
wl_stepsize(thinned.spct)
```

`times-.generic_spct` *Arithmetic Operators*

Description

Multiplication operator for spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 * e2
```

Arguments

e1	an object of class "generic_spct"
e2	an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct

`transmittance` *Transmittance*

Description

Summary transmittance for supplied wavebands from filter or object spectrum.

Usage

```
transmittance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## Default S3 method:
transmittance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## S3 method for class 'filter_spct'
transmittance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
    ...
)

## S3 method for class 'object_spct'
transmittance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'filter_mspct'
transmittance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx"
)

## S3 method for class 'object_mspct'
transmittance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an R object.
`w.band`	waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.

`quantity`	character string One of `"average"` or `"mean"`, `"total"`, `"contribution"`, `"contribution.pc"`, `"relative"` or `"relative.pc"`.
`wb.trim`	logical if `TRUE` wavebands crossing spectral data boundaries are trimmed, if `FALSE`, they are discarded.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`...`	ignored (possibly used by derived methods).
`naming`	character one of `"long"`, `"default"`, `"short"` or `"none"`. Used to select the type of names to assign to returned value.
`attr2tb`	character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
`idx`	character Name of the column with the names of the members of the collection of spectra.
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named `numeric` vector in the case of methods for individual spectra, with one value for each `waveband` passed to parameter `w.band`. A `data.frame` in the case of collections of spectra, containing one column for each `waveband` object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- `transmittance(default)`: Default method
- `transmittance(filter_spct)`: Method for filter spectra
- `transmittance(object_spct)`: Method for object spectra
- `transmittance(filter_mspct)`: Calculates transmittance from a `filter_mspct`
- `transmittance(object_mspct)`: Calculates transmittance from a `object_mspct`

Note

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

```
transmittance(polyester.spct, waveband(c(280, 315)))
transmittance(polyester.spct, waveband(c(315, 400)))
transmittance(polyester.spct, waveband(c(400, 700)))
```

Trig *Trigonometric Functions*

Description

Trigonometric functions for object of `generic_spct` and derived classes. \ The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of `x` and the current value of output options.

Usage

```
## S3 method for class 'generic_spct'
cos(x)

## S3 method for class 'generic_spct'
sin(x)

## S3 method for class 'generic_spct'
tan(x)

## S3 method for class 'generic_spct'
acos(x)

## S3 method for class 'generic_spct'
asin(x)

## S3 method for class 'generic_spct'
atan(x)
```

Arguments

`x`	an object of class "generic_spct" or a derived class.

trimInstrDesc *Trim the "instr.desc" attribute*

Description

Function to trim the "instr.desc" attribute of a generic_spct or a summary_generic_spct object, by default discarding all fields except for spectrometer.name, spectrometer.sn, bench.grating, bench.slit, and entrance.optics.

Usage

```
trimInstrDesc(
  x,
  fields = c("time", "spectrometer.name", "spectrometer.sn", "bench.grating",
    "bench.slit", "entrance.optics")
)
```

Arguments

x	a generic_spct object or a summary_generic_spct object.
fields	a character vector with the names of the fields to keep, or if first member is "-", the names of fields to delete; "*" as the first member of the vector makes the function a no-op, leaving the spectrum object unaltered.

Details

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object or a summary_generic_spct object, or if the "instr.desc" attribute is not present in a generic_spct object, x is not modified.

Attempts to remove or keep fields that are not present in the attribute are ignored silently. The value of fields in the attribute is never modified, fields are either kept unchanged or removed.

Value

x, possibly with the "instr.desc" attribute modified.

Note

Some of the spectrometer-specific metadata can be large, as they can include calibration coefficients. In the case of R package 'ooacquire' also pointers to Java objects may need to be deleted.

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrSettings()

Examples

```
my.spct <- white_led.cps_spct
names(instr_descriptor(my.spct))
trimInstrDesc(my.spct) # modified by reference!
names(instr_descriptor(my.spct))
```

---

trimInstrSettings *Trim the "instr.settings" attribute*

Description

Trim the "instr.settings" attribute of an existing generic_spct object or of a summary_generic_spct object, by discarding some fields.

Usage

```
trimInstrSettings(x, fields = "*")
```

Arguments

x	a generic_spct object or a summary_generic_spct object.
fields	a character vector with the names of the fields to keep, or if first member is "-", the names of fields to delete; "*" as first member of the vector makes the function a no-op, leaving the spectrum object unaltered.

Details

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object or a summary_generic_spct object, or if the "instr.settings" attribute is not present in x, x is not modified.

Attempts to remove or keep fields that are not present in the attribute are ignored silently. The value of fields in the attribute is never modified, fields are either kept unchanged or removed.

Value

x, possibly with the "instr.settings" attribute modified.

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(), getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(), setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(), subset_attributes(), trimInstrDesc()

Examples

```
my.spct <- white_led.cps_spct
names(instr_settings(my.spct))
trimInstrSettings(my.spct, fields = c("-", "pix.selector")) # by reference!
names(instr_settings(my.spct))
```

---

trim_spct *Trim (or expand) head and/or tail of a spectrum*

Description

Trim head and tail of a spectrum based on wavelength limits, interpolating the values at the boundaries of the range. Trimming is needed for example to remove short wavelength noise when the measured spectrum extends beyond the known emission spectrum of the measured light source. Occasionally one may want also to expand the wavelength range.

Usage

```
trim_spct(
  spct,
  range = NULL,
  low.limit = NULL,
  high.limit = NULL,
  use.hinges = TRUE,
  fill = NULL,
  byref = FALSE,
  verbose = getOption("photobiology.verbose")
)

trim_mspct(
  mspct,
  range = NULL,
  low.limit = NULL,
  high.limit = NULL,
  use.hinges = TRUE,
  fill = NULL,
  byref = FALSE,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)

trim2overlap(
  mspct,
  use.hinges = TRUE,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)

extend2extremes(
  mspct,
  use.hinges = TRUE,
  fill = NA,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`spct`	an object of class "generic_spct".
`range`	a numeric vector of length two, or any other object for which method range() will return a numeric vector of length two.
`low.limit`	shortest wavelength to be kept (defaults to shortest w.length value).
`high.limit`	longest wavelength to be kept (defaults to longest w.length value).
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`fill`	if fill==NULL then tails are deleted, otherwise tails or s.irrad are filled with the value of fill.
`byref`	logical indicating if new object will be created by reference or by copy of spct.
`verbose`	logical.
`mspct`	an object of class "generic_mspct"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

a spectrum object or a collection of spectral objects of the same class as `x` with wavelength heads and tails clipped or extended.

Note

When expanding a spectrum, if fill==NULL, then expansion is not performed. Range can be "waveband" object, a numeric vector or a list of numeric vectors, or any other user-defined or built-in object for which `range()` returns a numeric vector of length two, that can be interpreted as wavelengths expressed in nm.

See Also

Other trim functions: clip_wl(), trim_waveband(), trim_wl()

Examples

```
trim_spct(sun.spct, low.limit=300)
trim_spct(sun.spct, low.limit=300, fill=NULL)
trim_spct(sun.spct, low.limit=300, fill=NA)
trim_spct(sun.spct, low.limit=300, fill=0.0)
trim_spct(sun.spct, range = c(300, 400))
trim_spct(sun.spct, range = c(300, NA))
trim_spct(sun.spct, range = c(NA, 400))
```

trim_tails *Trim (or expand) head and/or tail*

Description

Trim tails of a spectrum based on wavelength limits, interpolating the values at the boundaries.Trimming is needed for example to remove short wavelength noise when the measured spectrum extends beyond the known emission spectrum of the measured light source. Occasionally one may want also to expand the wavelength range.

Usage

```
trim_tails(
  x,
  y,
  low.limit = min(x),
  high.limit = max(x),
  use.hinges = TRUE,
  fill = NULL,
  verbose = TRUE
)
```

Arguments

x	numeric vector of wavelengths.
y	numeric vector of values for a spectral quantity.
low.limit	smallest x-value to be kept (defaults to smallest x-value in input).
high.limit	largest x-value to be kept (defaults to largest x-value in input).
use.hinges	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
fill	if fill == NULL then tails are deleted, otherwise tails of y are filled with the value of fill.
verbose	logical Use to suppress warnings.

Value

A data.frame with variables x and y.

Note

When expanding a spectrum, if `fill` == `NULL`, expansion is not performed with a warning.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```
head(sun.data)
head(with(sun.data,
     trim_tails(w.length, s.e.irrad, low.limit=300)))
head(with(sun.data,
     trim_tails(w.length, s.e.irrad, low.limit=300, fill=NULL)))
```

`trim_waveband` *Trim (or expand) head and/or tail*

Description

Trimming of waveband boundaries can be needed when the spectral data do not cover the whole waveband, or wavebands may have to be removed altogether.

Usage

```
trim_waveband(
  w.band,
  range = NULL,
  low.limit = 0,
  high.limit = Inf,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = TRUE,
 trunc.labels = getOption("photobiology.brief.trunc.names", default = c("]", "["))
)
```

Arguments

`w.band`	an object of class "waveband" or a list of such objects.
`range`	a numeric vector of length two, or any other object for which function range() will return a numeric vector of two wavelengths (nm).
`low.limit`	shortest wavelength to be kept (defaults to 0 nm).
`high.limit`	longest wavelength to be kept (defaults to Inf nm).
`trim`	logical (default is TRUE which trims the wavebands at the boundary, while FALSE discards wavebands that are partly off-boundary).
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`trunc.labels`	character vector of length one or two. The first string will be prepended to the waveband name and label on left truncation and the second appended on right truncation. If the vector is of length one, the same string will be used in both cases.

Details

This function will accept both individual wavebands or list of wavebands. When the input is a list, wavebands outside the range of the range will be removed from the list, and those partly outside the target range either "trimmed" to this edge truncated if `trim = TRUE` is passed or excluded if `trim = FALSE`). Waveband objects contain a name and a label that are used to label the returned values of calculations that make use of them. When a waveband object is truncated so that the definition changes, the name and label are also modified so that the change is visible when they are used. The name and label have a string prepended or appended, and what strings are used can be set with an R option.

Value

The returned value is a waveband object or a list of waveband objects depending on whether a single waveband object or a list of waveband objects was supplied as argument to formal parameter `w.band`. If no waveband is retained, in the first case, a NULL waveband object is returned, and in the second case, a list of length zero is returned. If the input is a named, list, names are preserved in the returned list.

Note

Modification of the name and label stored in the wavebands passed as input is done so that summaries produced with the modified objects can be recognized as different from those computed using the original definitions when the waveband objects are used. When the input is a named list, the names of the retained members of the list are not modified as these are not part of the definitions.

See Also

Other trim functions: `clip_wl()`, `trim_spct()`, `trim_wl()`

Examples

```
VIS <- waveband(c(380, 760)) # manometers

trim_waveband(VIS, c(400,700))
trim_waveband(VIS, low.limit = 400)
trim_waveband(VIS, high.limit = 700)
trim_waveband(VIS, c(400,700), trunc.labels = c(">", "<"))
trim_waveband(VIS, c(400,700), trunc.labels = "!")
```

---

trim_wl *Trim head and/or tail of a spectrum*

Description

Trim head and tail of a spectrum based on wavelength limits, with interpolation at range boundaries used by default. Expansion is also possible.

Usage

```
trim_wl(x, range, use.hinges, fill, ...)

## Default S3 method:
trim_wl(x, range, use.hinges, fill, ...)

## S3 method for class 'generic_spct'
trim_wl(x, range = NULL, use.hinges = TRUE, fill = NULL, ...)

## S3 method for class 'generic_mspct'
trim_wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'waveband'
trim_wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  ...
)

## S3 method for class 'list'
trim_wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  ...
)
```

Arguments

`x`	an R object.
`range`	a numeric vector of length two, or any other object for which function range() will return two.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`fill`	if `fill == NULL` then tails are deleted, otherwise tails are filled with the value of fill.
`...`	ignored (possibly used by derived methods).
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
`trim`	logical (default is TRUE which trims the wavebands at the boundary, while FALSE discards wavebands that are partly off-boundary).

Value

A copy of `x`, usually trimmed or expanded to a different length, either shorter or longer. Possibly with some of the original spectral data values replaced with `fill`.

Methods (by class)

- `trim_wl(default)`: Default for generic function
- `trim_wl(generic_spct)`: Trim an object of class "generic_spct" or derived.
- `trim_wl(generic_mspct)`: Trim an object of class "generic_mspct" or derived.
- `trim_wl(waveband)`: Trim an object of class "waveband".
- `trim_wl(list)`: Trim a list (of "waveband" objects).

Note

By default the `w.length` values for the first and last rows in the returned object are the values supplied as `range`.

trim_wl when applied to waveband objects always inserts hinges when trimming.

trim_wl when applied to waveband objects always inserts hinges when trimming.

See Also

Other trim functions: `clip_wl()`, `trim_spct()`, `trim_waveband()`

Examples

```
trim_wl(sun.spct, range = c(400, 500))
trim_wl(sun.spct, range = c(NA, 500))
trim_wl(sun.spct, range = c(400, NA))

trim_wl(sun_evening.spct)
trim_wl(sun_evening.mspct)
```

`two_filters.spct` *Transmittance spectrum of plastic films*

Description

Datasets containing the wavelengths at a 1 nm interval and fractional total transmittance for a clear polyester film and a yellow theatrical "gel".

Usage

```
two_filters.spct

two_filters.mspct

polyester.spct

yellow_gel.spct
```

Format

A `filter_spct` object with 611 rows and 2 variables. Individually as `filter_spct` objects, and together as a collection stored in a `filter_mspct` object and in a long-form `filter_spct` object.

An object of class `filter_mspct` (inherits from `generic_mspct`, `list`) with 2 rows and 1 columns.

An object of class `filter_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 561 rows and 2 columns.

An object of class `filter_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 611 rows and 2 columns.

Details

- `w.length` (nm).
- `Tfr` (0..1).
- `spct.idx` (names, only in `two_filters.spct`).

Note

Package 'photobiologyFilters' contains data sets for hundreds of optical filters and materials in objects of these same classes, ready to be used with package 'photobiology'.

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_sensors.mspct`, `water.spct`, `white_led.source_spct`

Examples

```
polyester.spct
yellow_gel.spct
summary(two_filters.mspct)
```

`two_sensors.mspct` *Spectral response of two light sensors.*

Description

A dataset containing a collection of two spectra.

Usage

```
two_sensors.mspct

two_sensors.spct
```

Format

A `response_spct` object with 186 rows and 2 variables

An object of class `response_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 280 rows and 4 columns.

Details

The spectra in `photodiode.spct` and `ccd.spct` stored as a collection in a `response_mspct` object named `response.mspct` with members `photodiode` and `ccd`, and and in long form in a `link{response_spct}` object named `response.mspct` identified bit the levels of factor `spct.idx`.

See Also

photodiode.spct and ccd.spct.

Other Spectral data examples: A.illuminant.spct, D50.illuminant.spct, D65.illuminant.spct, Ler_leaf.spct, black_body.spct, ccd.spct, clear.spct, filter_cps.mspct, green_leaf.spct, phenylalanine.spct, photodiode.spct, sun.spct, sun_daily.spct, sun_evening.spct, two_filters.spct, water.spct, white_led.source_spct

Examples

```
two_sensors.mspct
two_sensors.spct
```

---

uncollect2spct *Extract all members from a collection*

Description

Extract all members from a collection into separate objects in the parent frame of the call.

Usage

```
uncollect2spct(x, ...)

## Default S3 method:
uncollect2spct(x, ...)

## S3 method for class 'generic_mspct'
uncollect2spct(
  x,
  name.tag = ".spct",
  ignore.case = FALSE,
  check.names = TRUE,
  check.overwrite = TRUE,
  ...
)
```

Arguments

x	An R object
...	additional named arguments passed down to f.
name.tag	character. A string used as tag for the names of the objects. If of length zero, names of members are used as named of objects. Otherwise the tag is appended, unless already present in the member name.
ignore.case	logical. If FALSE, the pattern matching used for name.tag is case sensitive and if TRUE, case is ignored during matching.

`check.names` logical. If TRUE then the names of the objects created are checked to ensure that they are syntactically valid variable names and unique. If necessary they are adjusted (by make.names) so that they are, and if FALSE names are used as is.

`check.overwrite` logical. If TRUE trigger an error if an exisitng object would be overwritten, and if FALSE silently overwrite objects.

Value

Utility used for its side effects, invisibly returns a character vector with the names of the objects created.

Methods (by class)

- `uncollect2spct(default)`: Default for generic function
- `uncollect2spct(generic_mspct)`:

See Also

Other experimental utility functions: `collect2mspct()`, `drop_user_cols()`, `thin_wl()`

Examples

```
my.mscpt <- source_mspct(list(sun1.spct = sun.spct, sun2.spct = sun.spct))
uncollect2spct(my.mscpt)
ls(pattern = "*.spct")
```

`untag` *Remove tags*

Description

Remove tags from an R object if present, otherwise return the object unchanged.

Usage

```
untag(x, ...)

## Default S3 method:
untag(x, ...)

## S3 method for class 'generic_spct'
untag(x, byref = FALSE, ...)

## S3 method for class 'generic_mspct'
untag(x, byref = FALSE, ...)
```

Arguments

`x`	an R object.
`...`	ignored (possibly used by derived methods).
`byref`	logical indicating if new object will be created by reference or by copy of x

Value

if `x` contains tag data they are removed and the "spct.tags" attribute is set to `NA`, while if `x` has no tags, it is not modified. In either case, the byref argument is respected: in all cases if `byref = FALSE` a copy of `x` is returned.

Methods (by class)

- `untag(default)`: Default for generic function
- `untag(generic_spct)`: Specialization for generic_spct
- `untag(generic_mspct)`: Specialization for generic_spct

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `wb2rect_spct()`, `wb2spct()`, `wb2tagged_spct()`

`upgrade_spct` *Upgrade one spectral object*

Description

Update the spectral class names of objects to those used in photobiology (>= 0.6.0) and add 'version' attribute as used in photobiology (>= 0.70).

Usage

```
upgrade_spct(object)
```

Arguments

`object`	generic.spct A single object to upgrade

Value

The modified object (invisibly).

Note

The object is modified by reference. The class names with ending ".spct" replaced by their new equivalents ending in "_spct".

See Also

Other upgrade from earlier versions: `is.old_spct()`, `upgrade_spectra()`

`upgrade_spectra` *Upgrade one or more spectral objects*

Description

Update the spectral class names of objects to those used in photobiology (>= 0.6.0).

Usage

```
upgrade_spectra(obj.names = ls(parent.frame()))
```

Arguments

`obj.names`	char Names of objects to upgrade as a vector of character strings

Value

The modified object (invisibly).

Note

The objects are modified by reference. The class names with ending ".spct" are replaced by their new equivalents ending in "_spct". `object.names` can safely include names of any R object. Names of objects which do not belong to any the old `.spct` classes are ignored. This makes it possible to supply as argument the output from `ls`, the default, or its equivalent `objects`.

See Also

Other upgrade from earlier versions: `is.old_spct()`, `upgrade_spct()`

`using_Tfr` *Use photobiology options*

Description

Execute an R expression, possibly compound, using a certain setting for spectral data related options.

Usage

```
using_Tfr(expr)

using_Afr(expr)

using_A(expr)

using_energy(expr)

using_photon(expr)

using_quantum(expr)
```

Arguments

`expr`	an R expression to execute.

Value

The value returned by the execution of `expression`.

References

Based on `withOptions()` as offered by Thomas Lumley, and listed in https://www.burns-stat.com/the-options-mechanism-in-r/, section Deep End, of "The Options mechanism in R" by Patrick Burns.

valleys *Valleys or local minima*

Description

Function that returns a subset of an R object with observations corresponding to local maxima.

Usage

```
valleys(
  x,
  span,
  global.threshold,
  local.threshold,
  local.reference,
  threshold.range,
  strict,
  ...
)
```

```
## Default S3 method:
valleys(
  x,
  span,
  global.threshold = NA,
  local.threshold = NA,
  local.reference = NA,
  threshold.range = NA,
  strict,
  ...
)

## S3 method for class 'numeric'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'data.frame'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  x.var.name = NULL,
  y.var.name = NULL,
  var.name = y.var.name,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'source_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "median",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'response_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'filter_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'reflector_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'solute_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'cps_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_spct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'source_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

```
## S3 method for class 'reflector_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
valleys(
  x,
  span = 5,
  global.threshold = NULL,
  local.threshold = NULL,
  local.reference = "minimum",
  threshold.range = NULL,
  strict = FALSE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	numeric vector.
`span`	odd positive integer A peak is defined as an element in a sequence which is greater than all other elements within a moving window of width `span` centred at that element. The default value is 5, meaning that a peak is taller than its four nearest neighbours. `span = NULL` extends the span to the whole length of `x`.
`global.threshold`	numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height or depth expressed in data units. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as relative to `threshold.range`. In both cases it sets a *global* height (depth) threshold below which peaks (valleys) are ignored. A bare negative `numeric` value indicates the *global* height (depth) threshold below which peaks (valleys) are be ignored. If `global.threshold = NULL`, no threshold is applied and all peaks returned.
`local.threshold`	numeric A value belonging to class `"AsIs"` is interpreted as an absolute minimum height (depth) expressed in data units relative to a within-window computed reference value. A bare `numeric` value (normally between 0.0 and 1.0), is interpreted as expressed in units relative to `threshold.range`. In both cases `local.threshold` sets a *local* height (depth) threshold below which peaks (valleys) are ignored. If `local.threshold = NULL` or if `span` spans the whole of `x`, no threshold is applied.
`local.reference`	character One of `"median"`, `"median.log"`, `"median.sqrt"`, `"farthest"`, `"farthest.log"`

	or "farthest.sqrt". The reference used to assess the height of the peak, either the minimum/maximum value within the window or the median of all values in the window.
threshold.range	numeric vector If of length 2 or a longer vector range(threshold.range) is used to scale both thresholds. With NULL, the default, range(x) is used, and with a vector of length one range(threshold.range, x) is used, i.e., the range is expanded.
strict	logical flag: if TRUE, an element must be strictly greater than all other values in its window to be considered a peak.
...	ignored
na.rm	logical indicating whether NA values should be stripped before searching for peaks.
var.name, x.var.name, y.var.name	character Name of column where to look for valleys.
refine.wl	logical Flag indicating if valley location should be refined by fitting a function.
method	character String with the name of a method. Currently only spline interpolation is implemented.
unit.out	character One of "energy" or "photon"
filter.qty	character One of "transmittance" or "absorbance"
.parallel	if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

As find_valleys, peaks and valleys call find_peaks to search for peaks and valleys, this explanation applies to the four functions. It also applies to stat_peaks and stat_valleys. Function find_peaks is a wrapper built onto function peaks from **splus2R**, adds support for peak height thresholds and handles span = NULL and non-finite (including NA) values differently than splus2R::peaks. Instead of giving an error when na.rm = FALSE and x contains NA values, NA values are replaced with the smallest finite value in x. span = NULL is treated as a special case and selects max(x). Passing strict = TRUE ensures that non-unique global and within window maxima are ignored, and can result in no peaks being returned.

Two tests make it possible to ignore irrelevant peaks. One test (global.threshold) is based on the absolute height of the peaks and can be used in all cases to ignore globally low peaks. A second test (local.threshold) is available when the window defined by ‘span‘ does not include all observations and can be used to ignore peaks that are not locally prominent. In this second approach the height of each peak is compared to a summary computed from other values within the window of width equal to span where it was found. In this second case, the reference value used within each window containing a peak is given by the argument passed to local.reference. Parameter threshold.range determines how the values passed as argument to global.threshold and local.threshold are scaled. The default, NULL uses the range of x. Thresholds for ignoring too small peaks are applied after peaks are searched for, and threshold values can in some cases result in no peaks being returned.

The `local.threshold` argument is used *as is* when `local.reference` is `"median"` or `"farthest"`, i.e., the same distance between peak and reference is used as cut-off irrespective of the value of the reference. In cases when the prominence of peaks is positively correlated with the baseline, a `local.threshold` that increases together with increasing computed within window median or farthest value applies apply a less stringent height requirement in regions with overall low height. In this case, natural logarithm or square root weighting can be requested with `local.reference` arguments `"median.log"`, `"farthest.log"`, `"median.sqrt"`, and `"farthest.sqrt"` as arguments for `local.reference`.

While functions `find_peaks` and `find_valleys` accept as input a `numeric` vector and return a `logical` vector, methods `peaks` and `valleys` accept as input different R objects, including spectra and collections of spectra and return a subset of the object. These methods are implemented using calls to functions `find_peaks`, `find_valleys` and `fit_peaks`.

Value

A subset of `x` with rows corresponding to local minima or global minimum.

Note

The default for parameter `strict` is `FALSE` in functions `find_peaks` and `find_valleys`, while the default in `peaks` is `strict = TRUE`.

See Also

Other peaks and valleys functions: `find_peaks()`, `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `wls_at_target()`

Examples

```
# default span = 5
valleys(sun.spct)
# global minimum
valleys(sun.spct, span = NULL)
valleys(sun.spct, span = NULL, strict = FALSE)
# a wider window
valleys(sun.spct, span = 51)
# global threshold relative to the range of s.e.irrad values
valleys(sun.spct, global.threshold = -0.2)
# global threshold in actual s.e.irrad values
valleys(sun.spct, global.threshold = -0.2, threshold.range = c(0, 1))
# local threshold  relative to the range of s.e.irrad values
valleys(sun.spct, local.threshold = 0.1)
# local threshold in actual s.e.irrad values
valleys(sun.spct, local.threshold = 0.1, threshold.range = c(0, 1))
# local threshold  relative to the range of s.e.irrad values, using window
# median instead of window minimum
valleys(sun.spct, local.threshold = 0.1, local.reference = "median")
# minimum, the default.
```

```
valleys(sun.spct, local.threshold = 0.1, local.reference = "farthest")

valleys(sun.spct)
```

---

verbose_as_default *Set error reporting options*

Description

Set error reporting related options easily.

Usage

```
verbose_as_default(flag = TRUE)

strict_range_as_default(flag = TRUE)
```

Arguments

flag	logical.

Value

Previous value of the modified option.

---

v_insert_hinges *Insert spectral data values at new wavelength values.*

Description

Inserting wavelengths values immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data have a relatively large wavelength step size and/or when the weighting function used has discontinuities in its value or slope. This function differs from `insert_hinges()` in that it returns a vector of `y` values instead of a `tibble`.

Usage

```
v_insert_hinges(x, y, h)
```

Arguments

x	numeric vector (sorted in increasing order).
y	numeric vector.
h	a numeric vector giving the wavelengths at which the `y` values should be inserted by interpolation, no interpolation is indicated by an empty numeric vector (`numeric(0)`).

Value

A numeric vector with the numeric values of y, but longer. Unless the hinge values were already present in y, each inserted hinge, expands the vector by two values.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_replace_hinges()`

`v_replace_hinges` *Overwrite spectral data values at existing wavelength values.*

Description

Overwriting spectral data with interpolated values at wavelengths values containing bad data is needed when cleaning spectral data. This function differs from `insert_hinges()` in that it returns a vector of y values instead of a `tibble`.

Usage

```
v_replace_hinges(x, y, h)
```

Arguments

x	numeric vector (sorted in increasing order).
y	numeric vector.
h	a numeric vector giving the wavelengths at which the y values should be replaced by interpolation, no interpolation is indicated by an empty numeric vector (`numeric(0)`).

Value

A numeric vector with the numeric values of y with values at the hinges replaced by interpolation of neighbours.

See Also

Other low-level functions operating on numeric vectors.: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`

`water.spct` *Molar spectral attenuation coefficient of water*

Description

A dataset containing the wavelengths at a 2 nm interval and the corresponding attenuation coefficients.

Usage

```
water.spct
```

Format

A `solute_spct` object with 251 rows and 2 variables

Details

- w.length (nm), range 300 to 800 nm.
- K.mole (cm-1/M)

Author(s)

Buiteveld et al. (1994) (original data)

References

H. Buiteveld and J. M. H. Hakvoort and M. Donze (1994) "The optical properties of pure water," in SPIE Proceedings on Ocean Optics XII, edited by J. S. Jaffe, 2258, 174–183.

`https://omlc.org/spectra/water/`

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `white_led.source_spct`

Examples

```
head(water.spct)
summary(water.spct)
solute_properties(water.spct)
cat(comment(water.spct))
```

waveband *Waveband constructor method*

Description

Constructor for "waveband" objects that can be used as input when calculating irradiances.

Usage

```
waveband(
  x = NULL,
  weight = NULL,
  SWF.e.fun = NULL,
  SWF.q.fun = NULL,
  norm = NULL,
  SWF.norm = NULL,
  hinges = NULL,
  wb.name = NULL,
  wb.label = wb.name
)

new_waveband(
  w.low,
  w.high,
  weight = NULL,
  SWF.e.fun = NULL,
  SWF.q.fun = NULL,
  norm = NULL,
  SWF.norm = NULL,
  hinges = NULL,
  wb.name = NULL,
  wb.label = wb.name
)
```

Arguments

`x`	any R object on which applying the method `range()` yields an vector of two numeric values, describing a range of wavelengths $[nm]$.
`weight`	a character string `"SWF"` or `"BSWF"`, use `NULL` (the default) to indicate no weighting used when calculating irradiance.
`SWF.e.fun, SWF.q.fun`	a functions giving multipliers for a spectral weighting function (energy and quantum, respectively) as a function of wavelength $[nm]$.
`norm`	a single numeric value indicating the wavelength $[nm]$ at which the SWF should be normalized to 1.0; `NULL` is interpreted as no normalization.

`SWF.norm`	a numeric value giving the native normalization wavelength $[nm]$ used by `SWF.e.fun` and `SWF.q.fun`.
`hinges`	a numeric vector giving the wavelengths at which values in `s.irrad` should be inserted by interpolation before integration is attempted. No interpolation is indicated by an empty vector (`numeric(0)`), while interpolation at both boundaries of the band is indicated by `NULL`.
`wb.name`	character string giving the name for the waveband defined, default is `NULL` for an automatically generated name.
`wb.label`	character string giving the label of the waveband to be used for labelling computed summaries or plots, default is `wb.name`. (This is usually a shorter character string than `wb.name`.)
`w.low`, `w.high`	numeric value, wavelengths at the short end and long ends of the wavelength band $[nm]$.

Details

Objects of class `waveband` are used to store the different bits of information needed to compute summaries from spectral data by integration over wavelengths. The wavelength ranges, possible spectral weighting functions (SWF) or biological spectral weighting functions (BSWF), their normalization wavelengths and names and labels used for reporting the results are all stored in waveband objects. This facilitates the use of functions that compute summaries, as well as ensures consistency in computations and labelling, as all the bits of information are passed together. Class `"waveband"` is derived from R class `list`.

Value

a `waveband` object

Functions

- `new_waveband()`: A less flexible variant

See Also

Other waveband constructors: `split_bands()`

Examples

```
waveband(c(400,700))

new_waveband(400,700)
```

waveband_ratio *Photon or energy ratio*

Description

This function gives the (energy or photon) irradiance ratio between two given wavebands of a radiation spectrum.

Usage

```
waveband_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.out.num = NULL,
  unit.out.denom = unit.out.num,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

`w.length`	numeric Vector of wavelengths [nm].
`s.irrad`	numeric vector of spectral irradiances in [$W\,m^{-2}\,nm^{-1}$] or [$mol\,s^{-1}\,sm^{-2}\,nm^{-1}$] as indicated by the argument pased to `unit.in`.
`w.band.num`, `w.band.denom`	waveband objects used to compute the numerator and denominator of the ratio.
`unit.out.num`, `unit.out.denom`	character Base of expression used to compute the numerator and denominator of the ratio. Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`unit.in`	character Allowed values `"energy"`, and `"photon"`, or its alias `"quantum"`.
`check.spectrum`	logical Flag indicating whether to sanity check input data, default is TRUE.
`use.cached.mult`	logical Flag indicating whether multiplier values should be cached between calls.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a single numeric value giving the ratio

Note

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`. From version 0.9.19 onwards use of this default does not trigger a warning, but instead is used silently.

Examples

```
# photon:photon ratio
with(sun.data,
     waveband_ratio(w.length, s.e.irrad,
                    new_waveband(400,500),
                    new_waveband(400,700), "photon"))
# energy:energy ratio
with(sun.data,
     waveband_ratio(w.length, s.e.irrad,
                    new_waveband(400,500),
                    new_waveband(400,700), "energy"))
# energy:photon ratio
with(sun.data,
     waveband_ratio(w.length, s.e.irrad,
                    new_waveband(400,700),
                    new_waveband(400,700),
                    "energy", "photon"))
# photon:photon ratio waveband : whole spectrum
with(sun.data,
     waveband_ratio(w.length, s.e.irrad,
                    new_waveband(400,500),
                    unit.out.num="photon"))
# photon:photon ratio of whole spectrum should be equal to 1.0
with(sun.data,
     waveband_ratio(w.length, s.e.irrad,
     unit.out.num="photon"))
```

`wb2rect_spct` *Create tagged spectrum from wavebands*

Description

Create a generic_spct object with wavelengths from the range of wavebands in a list. The spectrum is suitable for plotting labels, symbols, rectangles or similar, as the midpoint of each waveband is added to the spectrum.

Usage

```
wb2rect_spct(w.band, short.names = TRUE, chroma.type = "CMF")

fast_wb2rect_spct(w.band, chroma.type = "CMF", simplify = TRUE)
```

Arguments

`w.band`	waveband or list of waveband objects The waveband(s) determine the wavelengths in variable `w.length` of the returned spectrum
`short.names`	logical Flag indicating whether to use short or long names for wavebands
`chroma.type`	character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class `chroma_spct` for any other trichromic visual system.
`simplify`	logical Flag indicating whether to merge neighboring rectangles of equal color. Simplification is done only for narrow wavebands.

Value

A `generic.spectrum` object, with columns w.length, wl.low, wl.hi, wl.color, wb.color and wb.name. The w.length values are the midpoint of the wavebands, wl.low and wl.high give the boundaries of the wavebands, wl.color the color definition corresponding to the wavelength at the center of the waveband and wb.color the color of the waveband as a whole (assuming a flat energy irradiance spectrum). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from `generic_spct`.

Note

Function `fast_wb2rect_spct()` differs from `wb2rect_spct()` in that it computes colors for narrow wavebands based on the midpoint wavelength and uses vectorization when possible. It always returns color definitions with short names, which are also used as waveband names for narrow wavebands and merged wavebands. The purpose of merging of rectangles is to speed up rendering and to reduce the size of vector graphics output. This function should be used with care as the color definitions returned are only approximate and original waveband names can be lost.

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `untag()`, `wb2spct()`, `wb2tagged_spct()`

---

wb2spct *Create spectrum from wavebands*

Description

Create a generic_spct object with wavelengths from wavebands in a list.

Usage

```
wb2spct(w.band)
```

Arguments

`w.band`	waveband or list of waveband objects The waveband(s) determine the wavelengths in variable `w.length` of the returned spectrum

Value

A generic.spectrum object, with columns w.length set to the *union* of all boundaries and hinges defined in the waveband(s). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from `generic_spct`.

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `untag()`, `wb2rect_spct()`, `wb2tagged_spct()`

`wb2tagged_spct` *Create tagged spectrum from wavebands*

Description

Create a tagged `generic_spct` object with wavelengths from the range of wavebands in a list, and names of the same bands as factor levels, and corresponding color definitions. The spectrum is not suitable for plotting labels, symbols, rectangles or similar, as the midpoint of each waveband is not added to the spectrum.

Usage

```
wb2tagged_spct(
  w.band,
  use.hinges = TRUE,
  short.names = TRUE,
  chroma.type = "CMF",
  ...
)
```

Arguments

`w.band`	waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are tagged and the wavelengths returned in variable `w.length`.
`use.hinges`	logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
`short.names`	logical Flag indicating whether to use short or long names for wavebands.
`chroma.type`	character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class `chroma_spct` for any other trichromic visual system.
`...`	ignored (possibly used by derived methods).

Value

A spectrum as returned by `wb2spct` but additionally tagged using function `tag`

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `untag()`, `wb2rect_spct()`, `wb2spct()`

`wb_trim_as_default` *Set computation options*

Description

Set computation related options easily.

Usage

```
wb_trim_as_default(flag = TRUE)

use_cached_mult_as_default(flag = TRUE)
```

Arguments

`flag`	logical.

Value

Previous value of the modified option.

`white_led.source_spct` *White led bulb spectrum*

Description

Datasets containing wavelengths and the corresponding spectral irradiance data for an Osram warm white led lamp, and the corresponding raw instrument counts and counts per second data underlying them.

Usage

```
white_led.source_spct

white_led.cps_spct

white_led.raw_spct
```

Format

A `source_spct` object with 1421 rows and 2 columns, a `cps_spct` object with 2068 rows and 2 columns, and a `raw_spct` object with 2068 rows and 4 columns.

An object of class `cps_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 2068 rows and 2 columns.

An object of class `raw_spct` (inherits from `generic_spct`, `tbl_df`, `tbl`, `data.frame`) with 2068 rows and 4 columns.

Details

- w.length (nm), range 250 to 900 nm.
- s.e.irrad (W m-2 nm-1)

or

- w.length (nm), range 188 to 1117 nm.
- cps

or

- w.length (nm), range 188 to 1117 nm.
- counts_1
- counts_2
- counts_3

See Also

Other Spectral data examples: `A.illuminant.spct`, `D50.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `filter_cps.mspct`, `green_leaf.spct`, `phenylalanine.spct`, `photodiode.spct`, `sun.spct`, `sun_daily.spct`, `sun_evening.spct`, `two_filters.spct`, `two_sensors.mspct`, `water.spct`

Examples

```
white_led.source_spct
```

`wl2wavenumber` *Wavelength conversions*

Description

Convert wavelength (nm) into wave number, frequency (Hz) or energy per photon (J, or eV) and back.

Usage

```
wl2wavenumber(w.length, unit.exponent = 0)

wavenumber2wl(wavenumber, unit.exponent = 0)

wl2frequency(w.length, unit.exponent = 0)

frequency2wl(frequency, unit.exponent = 0)

wl2energy(w.length, unit.exponent = 0, unit = "joule")

energy2wl(photon.energy, unit.exponent = 0, unit = "joule")
```

Arguments

`w.length`	numeric wavelength (nm)
`unit.exponent`	integer Exponent of the scale multiplier implicit in result, e.g., use 3 for kJ.
`wavenumber`	numeric Wave number in waves per metre, possibly with a scale factor according to `unit.exponent`.
`frequency`	numeric Frequency in Hz, possibly with the scale factor according to `unit.exponent`.
`unit`	character One of "joule" or "eV".
`photon.energy`	numeric Energy of one photon in joule or eV, possibly with a scale factor according to `unit.exponent`.

Details

These functions always expect as input and return wavelengths expressed in nanometres (nm) as all other functions in the R for photobiology suite of packages. Conversions depend on Plank's constant, *h*, the speed of light in vacuum, *c*, and Avogadro's number, N_A. The values used for these constants have at least nine significant digits.

Examples

```
wl2wavenumber(600) # wavelength in nm -> wave number
wavenumber2wl(1666666.66) # wave number -> wavelength in nm
wl2frequency(600) # wavelength in nm -> wave frequency (Hz)
frequency2wl(499654096666667) # wave frequency (Hz) -> wavelength in nm
wl2energy(600) # wavelength in nm -> energy of one photon (J)
wl2energy(600, unit = "eV") # wavelength in nm -> energy of one photon (eV)
wl2energy(600,
          unit.exponent = -3,
          unit = "eV")  # wavelength in nm -> energy of one photon (meV)
energy2wl(2066.40330,
          unit.exponent = -3,
          unit = "eV")  # energy of one photon (meV) -> wavelength (nm)
```

`wls_at_target` *Find wavelengths values corresponding to a target spectral value*

Description

Find wavelength values corresponding to a target spectral value in a spectrum. The name of the column of the spectral data to be used is inferred from the class of `x` and the argument passed to `unit.out` or `filter.qty` or their defaults that depend on R options set.

Usage

```
wls_at_target(x, target, interpolate, idfactor, na.rm, ...)

## Default S3 method:
wls_at_target(
  x,
  target = NULL,
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'data.frame'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  x.var.name = NULL,
  y.var.name = NULL,
  ...
)

## S3 method for class 'generic_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  col.name = NULL,
  y.var.name = col.name,
  ...
)

## S3 method for class 'source_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)
```

```
## S3 method for class 'response_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'solute_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'raw_spct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'generic_mspct'
wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
```

Arguments

`x`	data.frame or spectrum object.
`target`	numeric or character vector. A numeric value indicates the spectral quantity value for which wavelengths are to be searched. A character string representing a number is converted to numeric. A character value representing a number followed by a function name, will be also accepted and decoded, such that `"0.1max"` is interpreted as targeting one tenth of the maximum value in the column. The character strings "half.maximum" and "HM" are synonyms for "0.5max" while "half.range" and "HR" are synonyms for "0.5range".
`interpolate`	logical Indicating whether the nearest wavelength value in `x` should be returned or a value calculated by linear interpolation between wavelength values straddling the target.
`idfactor`	logical or character Generates an index column of factor type. If `idfactor = TRUE` then the column is auto named target.idx. Alternatively the column name can be directly passed as argument to `idfactor` as a character string.
`na.rm`	logical indicating whether `NA` values should be stripped before searching for the target.
`...`	currently ignored.
`x.var.name`, `y.var.name`, `col.name`	character The name of the columns in which to search for the target value. Use of `col.name` is deprecated, and is a synonym for `y.var.name`.

`unit.out`	character One of "energy" or "photon"
`filter.qty`	character One of "transmittance" or "absorbance"
`.parallel`	if TRUE, apply function in parallel, using parallel backend provided by foreach
`.paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A data.frame, a spectrum object or a collection of spectra object of the same class as `x` with fewer rows, possibly even no rows. If `FALSE` is passed to `interpolate` a subset of `x` is returned, otherwise a new object of the same class containing interpolated wavelengths for the `target` value is returned. As ‘target‘ accepts a vector or list as argument, a factor can be added to the output with the corresponding target value.

Note

When interpolation is used, only column `w.length` and the column against which the target value was compared are included in the returned object, otherwise, all columns in `x` are returned. We implement support for `data.frame` to simplify the coding of ’ggplot2’ stats using this function.

See Also

Other peaks and valleys functions: `find_peaks()`, `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `valleys()`

Examples

```
wls_at_target(sun.spct, target = 0.1)
wls_at_target(sun.spct, target = 2e-6, unit.out = "photon")
wls_at_target(polyester.spct, target = "HM")
wls_at_target(polyester.spct, target = "HM", interpolate = TRUE)
wls_at_target(polyester.spct, target = "HM", idfactor = "target")
wls_at_target(polyester.spct, target = "HM", filter.qty = "absorbance")
```

`wl_max` *Wavelength maximum*

Description

A method specialization that returns the wavelength maximum $[nm]$ from objects of classes `waveband` or of class `generic_spct` or derived.

Usage

```
wl_max(x, na.rm = FALSE)

## S3 method for class 'waveband'
max(..., na.rm = FALSE)

## S3 method for class 'generic_spct'
max(..., na.rm = FALSE)

## S3 method for class 'generic_mspct'
max(..., na.rm = FALSE, idx = "spct.idx")
```

Arguments

`x`	generic_spct, generic_mspct or waveband object.
`na.rm`	ignored
`...`	numeric, `waveband` or `generic_spct` arguments.
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a length-one vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- `max(generic_spct)`:
- `max(generic_mspct)`:

Examples

```
max(sun.spct)
wl_max(sun.spct)
```

`wl_midpoint` *Midpoint*

Description

A method that returns the wavelength $[nm]$ (or value) at the center of the wavelength range of objects of classes `waveband` or of class `generic_spct` or derived (or the midpoint from a `numeric` vector).

Usage

```
wl_midpoint(x, ...)

midpoint(x, ...)

## Default S3 method:
midpoint(x, ...)

## S3 method for class 'numeric'
midpoint(x, ...)

## S3 method for class 'waveband'
midpoint(x, ...)

## S3 method for class 'generic_spct'
midpoint(x, ...)

## S3 method for class 'generic_mspct'
midpoint(x, ..., idx = "spct.idx")
```

Arguments

`x`	an R object
`...`	not used in current version
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

A numeric value equal to `max(x) - min(x)) / 2`. In the case of spectral objects a wavelength $[nm]$. For any other R object, according to available definitions of `min` and `max`.

Methods (by class)

- `midpoint(default)`: Default method for generic function
- `midpoint(numeric)`: Default method for generic function
- `midpoint(waveband)`: Wavelength at center of a "waveband".
- `midpoint(generic_spct)`: Method for "generic_spct".
- `midpoint(generic_mspct)`: Method for "generic_mspct" objects.

See Also

Other wavelength summaries: `wl_min()`, `wl_range()`, `wl_stepsize()`

Other wavelength summaries: `wl_min()`, `wl_range()`, `wl_stepsize()`

Other wavelength summaries: `wl_min()`, `wl_range()`, `wl_stepsize()`

Examples

```
midpoint(10:20)
midpoint(sun.spct)
wl_midpoint(sun.spct)

midpoint(sun.spct)
```

`wl_min` *Wavelength minimum*

Description

A method specialization that returns the wavelength minimum [nm] from objects of classes `waveband` or of class `generic_spct` or derived.

Usage

```
wl_min(x, na.rm = FALSE)

## S3 method for class 'waveband'
min(..., na.rm = FALSE)

## S3 method for class 'generic_spct'
min(..., na.rm = FALSE)

## S3 method for class 'generic_mspct'
min(..., na.rm = FALSE, idx = "spct.idx")
```

Arguments

`x`	generic_spct, generic_mspct or waveband object.
`na.rm`	ignored
`...`	not used in current version
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a length-one vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- `min(generic_spct)`:
- `min(generic_mspct)`:

See Also

Other wavelength summaries: `wl_midpoint()`, `wl_range()`, `wl_stepsize()`

Examples

```
min(sun.spct)
wl_min(sun.spct)
```

`wl_range` *Wavelength range*

Description

A method specialization that returns the wavelength range $[nm]$ from objects of classes `waveband` or of class `generic_spct` or derived.

Usage

```
wl_range(x, na.rm = FALSE)

## S3 method for class 'waveband'
range(..., na.rm = FALSE)

## S3 method for class 'generic_spct'
range(..., na.rm = FALSE)

## S3 method for class 'generic_mspct'
range(..., na.rm = FALSE, idx = "spct.idx")
```

Arguments

`x`	generic_spct, generic_mspct or waveband object.
`na.rm`	ignored
`...`	a single R object
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

a length-two vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- `range(generic_spct)`:
- `range(generic_mspct)`:

See Also

Other wavelength summaries: `wl_midpoint()`, `wl_min()`, `wl_stepsize()`

Examples

```
range(sun.spct)
wl_range(sun.spct)

range(sun.spct)
```

`wl_stepsize` *Stepsize*

Description

Method returning the range of step sizes in an object; i.e., the Range of differences between successive sorted values. In particular the wavelength step sizes [nm] of objects of class `generic_spct` or derived (or the step sizes of values in a `numeric` vector).

Usage

```
wl_stepsize(x, ...)

stepsize(x, ...)

## Default S3 method:
stepsize(x, ...)

## S3 method for class 'numeric'
stepsize(x, ...)

## S3 method for class 'generic_spct'
stepsize(x, ...)

## S3 method for class 'generic_mspct'
stepsize(x, ..., idx = "spct.idx")
```

Arguments

`x`	an R object
`...`	not used in current version
`idx`	character Name of the column with the names of the members of the collection of spectra.

Value

A numeric vector of length 2 with min and maximum stepsize values.

Methods (by class)

- `stepsize(default)`: Default function usable on numeric vectors.
- `stepsize(numeric)`: Method for numeric vectors.
- `stepsize(generic_spct)`: Method for "generic_spct" objects.
- `stepsize(generic_mspct)`: Method for "generic_mspct" objects.

See Also

Other wavelength summaries: `wl_midpoint()`, `wl_min()`, `wl_range()`

Examples

```
stepsize(sun.spct)
wl_stepsize(sun.spct)

stepsize(sun.spct)
```

`w_length2rgb` *Wavelength to rgb color conversion*

Description

Calculates rgb values from spectra based on human color matching functions

Usage

```
w_length2rgb(w.length, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)
```

Arguments

`w.length`	numeric Vector of wavelengths $[nm]$.
`sens`	chroma_spct Used as chromaticity definition.
`color.name`	character Used for naming the rgb color definition.

Value

A vector of colors defined using `rgb()`. The numeric values of the RGB components can be obtained using function `col2rgb()`.

See Also

Other color functions: `rgb_spct()`, `w_length_range2rgb()`

Examples

```
col2rgb(w_length2rgb(580))
col2rgb(w_length2rgb(c(400, 500, 600, 700)))
col2rgb(w_length2rgb(c(400, 500, 600, 700), color.name=c("a","b","c","d")))
col2rgb(w_length2rgb(c(400, 500, 600, 700), color.name="a"))
```

`w_length_range2rgb` *Wavelength range to rgb color conversion*

Description

Calculates rgb values from spectra based on human color matching functions

Usage

```
w_length_range2rgb(
  w.length,
  sens = photobiology::ciexyzCMF2.spct,
  color.name = NULL
)
```

Arguments

`w.length`	numeric vector of wavelengths (nm) of length 2. If longer, its range is used.
`sens`	chroma_spct Used as the chromaticity definition.
`color.name`	character Used for naming the rgb color definition(s) returned.

Value

A vector of colors defined using `rgb()`. The numeric values of the RGB components can be obtained by calling function `col2rgb`.

See Also

Other color functions: `rgb_spct()`, `w_length2rgb()`

Examples

```
col2rgb(w_length_range2rgb(c(500,600)))
col2rgb(w_length_range2rgb(550))
col2rgb(w_length_range2rgb(500:600))
```

`^.generic_spct` *Arithmetic Operators*

Description

Power operator for spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 ^ e2
```

Arguments

`e1`	an object of class "generic_spct"
`e2`	a numeric vector. possibly of length one.

See Also

Other math operators and functions: `MathFun`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`

Index