Package ‘photobiology’

June 7, 2020

Type Package
Title Photobiological Calculations
Version 0.10.4
Date 2020-06-06

License GPL (>= 2)
Depends R (>= 3.6.0)
Imports stats, polynom (>= 1.4-0), tibble (>= 3.0.1), stringr (>= 1.4.0), lubridate (>= 1.7.8), plyr (>= 1.8.4), dplyr (>= 1.0.0), splus2R (>= 1.2-2), zoo (>= 1.8-7), rlang (>= 0.4.6)
Suggests knitr (>= 1.28), rmarkdown (>= 2.2), testthat (>= 2.3.2), roxygen2 (>= 7.1.0)

LazyLoad yes
LazyData yes
ByteCompile true

URL https://docs.r4photobiology.info/photobiology,
     https://bitbucket.org/aphalo/photobiology

BugReports https://bitbucket.org/aphalo/photobiology/issues

Encoding UTF-8
RoxygenNote 7.1.0
VignetteBuilder knitr
NeedsCompilation no
Author Pedro J. Aphalo [aut, cre] (<https://orcid.org/0000-0003-3385-972X>),
Titta K. Kotilainen [ctb] (<https://orcid.org/0000-0002-2822-9734>),
Glenn Davis [ctb],
Agnese Fazio [ctb]

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

Repository CRAN

Date/Publication 2020-06-07 05:40:05 UTC

R topics documented:

- photobiology-package ........................................ 8
- A.illuminant.spc ........................................... 10
- A2T ......................................................... 11
- absorbance .................................................. 12
- absorptance .................................................. 15
- add_attr2tb .................................................. 18
- Afr2T ......................................................... 19
- any2T ......................................................... 21
- as.calibration_mspct ....................................... 22
- as.calibration_spc ......................................... 24
- as.chroma_mspct ........................................... 25
- as.chroma_spc .............................................. 26
- as.cps_mspct ............................................... 27
- as.cps_spc ................................................. 28
- as.filter_mspct ........................................... 29
- as.filter_spc ............................................... 31
- as.generic_mspct .......................................... 32
- as.generic_spc ............................................ 34
- as.matrix_mspct ........................................... 34
- as.object_mspct ........................................... 35
- as.object_spc .............................................. 37
- as.raw_mspct ............................................... 38
- as.raw_spc .................................................. 39
- as.reflector_mspct ....................................... 40
- as.reflector_spc .......................................... 42
- as.response_mspct ....................................... 43
- as.response_spc .......................................... 44
- as.solar_date ............................................. 45
- as.source_mspct .......................................... 46
- as.source_spc ............................................. 48
- as_energy .................................................. 49
- as_quantum ................................................ 49
- as_quantum_mol ........................................... 50
- as_tod ....................................................... 51
- average_spc ............................................... 52
- beesxyzCMF.spc ........................................... 52
- black_body.spc ........................................... 53
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>54</td>
</tr>
<tr>
<td>calc_multipliers</td>
<td>54</td>
</tr>
<tr>
<td>calc_source_output</td>
<td>55</td>
</tr>
<tr>
<td>ccd.spct</td>
<td>56</td>
</tr>
<tr>
<td>checkTimeUnit</td>
<td>57</td>
</tr>
<tr>
<td>check_spct</td>
<td>58</td>
</tr>
<tr>
<td>check_spectrum</td>
<td>61</td>
</tr>
<tr>
<td>check_w.length</td>
<td>62</td>
</tr>
<tr>
<td>ciev10.spct</td>
<td>62</td>
</tr>
<tr>
<td>ciev2.spct</td>
<td>63</td>
</tr>
<tr>
<td>ciexyzCC10.spct</td>
<td>64</td>
</tr>
<tr>
<td>ciexyzCC2.spct</td>
<td>65</td>
</tr>
<tr>
<td>ciexyzCMF10.spct</td>
<td>66</td>
</tr>
<tr>
<td>ciexyzCMF2.spct</td>
<td>67</td>
</tr>
<tr>
<td>class_spct</td>
<td>68</td>
</tr>
<tr>
<td>clean</td>
<td>68</td>
</tr>
<tr>
<td>clear.spct</td>
<td>73</td>
</tr>
<tr>
<td>clear_body.spct</td>
<td>74</td>
</tr>
<tr>
<td>clip_wl</td>
<td>74</td>
</tr>
<tr>
<td>color_of</td>
<td>76</td>
</tr>
<tr>
<td>compare_spect</td>
<td>78</td>
</tr>
<tr>
<td>cone_fundamentals10.spct</td>
<td>79</td>
</tr>
<tr>
<td>convertTfrType</td>
<td>80</td>
</tr>
<tr>
<td>convertThickness</td>
<td>81</td>
</tr>
<tr>
<td>convertTimeUnit</td>
<td>82</td>
</tr>
<tr>
<td>convolve_each</td>
<td>83</td>
</tr>
<tr>
<td>copy_attributes</td>
<td>84</td>
</tr>
<tr>
<td>cps2irrad</td>
<td>85</td>
</tr>
<tr>
<td>D2.UV586</td>
<td>86</td>
</tr>
<tr>
<td>D2.UV653</td>
<td>86</td>
</tr>
<tr>
<td>D2.UV654</td>
<td>87</td>
</tr>
<tr>
<td>D2_spectrum</td>
<td>87</td>
</tr>
<tr>
<td>D65.illuminant.spct</td>
<td>88</td>
</tr>
<tr>
<td>day_night</td>
<td>89</td>
</tr>
<tr>
<td>defunct</td>
<td>92</td>
</tr>
<tr>
<td>despike</td>
<td>93</td>
</tr>
<tr>
<td>diffraction_single_slit</td>
<td>99</td>
</tr>
<tr>
<td>dim.generic_mspct</td>
<td>100</td>
</tr>
<tr>
<td>div-.generic_spct</td>
<td>101</td>
</tr>
<tr>
<td>div_spectra</td>
<td>101</td>
</tr>
<tr>
<td>e2q</td>
<td>103</td>
</tr>
<tr>
<td>e2qmol_multipliers</td>
<td>104</td>
</tr>
<tr>
<td>e2quantum_multipliers</td>
<td>104</td>
</tr>
<tr>
<td>enable_check_spct</td>
<td>105</td>
</tr>
<tr>
<td>energy_as_default</td>
<td>106</td>
</tr>
<tr>
<td>energy_irradiance</td>
<td>106</td>
</tr>
<tr>
<td>energy_ratio</td>
<td>107</td>
</tr>
<tr>
<td>eq_ratio</td>
<td>109</td>
</tr>
</tbody>
</table>
R topics documented:

Extract ................................................................. 111
Extract_mspct ......................................................... 113
e_fluence ............................................................... 114
e_irrad ................................................................. 117
e_ratio ................................................................. 120
e_response ............................................................ 123
FEL.BN.9101.165 ....................................................... 126
FEL_spectrum .......................................................... 126
findMultipleWl ........................................................ 127
find_peaks ............................................................. 127
find_spikes ............................................................ 128
find_wls ............................................................... 130
fit_peaks .............................................................. 131
fluence ............................................................... 133
format.solar_time ................................................... 136
format.tod_time ..................................................... 136
formatted_range .................................................... 137
fscale ................................................................. 137
fshift ................................................................. 142
generic_mspct ....................................................... 146
getBSWFUsed ........................................................ 148
getFilterProperties ................................................ 148
getHowMeasured .................................................... 150
getIdFactor .......................................................... 151
getInstrDesc ........................................................ 152
getInstrSettings ..................................................... 153
getMspctVersion ..................................................... 153
getMultipleWl ........................................................ 154
getNormalized ........................................................ 155
getRfrType ........................................................... 155
getScaled ............................................................. 156
getSpectVersion ..................................................... 157
getTfrType ............................................................ 157
g getTimeUnit .......................................................... 158
getWhatMeasured ...................................................... 159
getWhenMeasured ..................................................... 160
getWhereMeasured .................................................... 161
get_attributes ......................................................... 163
get_peaks .............................................................. 164
green_leaf.spekt ..................................................... 166
head_tail .............................................................. 167
insert_hinges ......................................................... 168
insert_spect_hinges ................................................ 169
integrate_spect ....................................................... 170
integrate_xy .......................................................... 171
interpolate_spect .................................................... 171
interpolate_spectrum ............................................... 173
interpolate_wl ........................................................ 174
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>irradi</td>
<td>175</td>
</tr>
<tr>
<td>irradiance</td>
<td>179</td>
</tr>
<tr>
<td>is.generic_mspct</td>
<td>180</td>
</tr>
<tr>
<td>is.generic_spct</td>
<td>181</td>
</tr>
<tr>
<td>is.old_spct</td>
<td>182</td>
</tr>
<tr>
<td>is.solar_time</td>
<td>183</td>
</tr>
<tr>
<td>is.summary_generic_spct</td>
<td>183</td>
</tr>
<tr>
<td>is.waveband</td>
<td>184</td>
</tr>
<tr>
<td>isValidInstrDesc</td>
<td>185</td>
</tr>
<tr>
<td>isValidInstrSettings</td>
<td>186</td>
</tr>
<tr>
<td>is_absorbance_based</td>
<td>186</td>
</tr>
<tr>
<td>is_effective</td>
<td>187</td>
</tr>
<tr>
<td>is_normalized</td>
<td>188</td>
</tr>
<tr>
<td>is_photon_based</td>
<td>189</td>
</tr>
<tr>
<td>is_scaled</td>
<td>190</td>
</tr>
<tr>
<td>is_tagged</td>
<td>191</td>
</tr>
<tr>
<td>join_mspct</td>
<td>191</td>
</tr>
<tr>
<td>labels</td>
<td>193</td>
</tr>
<tr>
<td>Ler_leaf.spct</td>
<td>194</td>
</tr>
<tr>
<td>Ler_leaf_rflt.spct</td>
<td>195</td>
</tr>
<tr>
<td>Ler_leaf_trns.spct</td>
<td>196</td>
</tr>
<tr>
<td>Ler_leaf_trns_i.spct</td>
<td>197</td>
</tr>
<tr>
<td>log</td>
<td>198</td>
</tr>
<tr>
<td>MathFun</td>
<td>199</td>
</tr>
<tr>
<td>merge2object_spct</td>
<td>199</td>
</tr>
<tr>
<td>merge_attributes</td>
<td>200</td>
</tr>
<tr>
<td>minus-.generic_spct</td>
<td>201</td>
</tr>
<tr>
<td>mod-.generic_spct</td>
<td>202</td>
</tr>
<tr>
<td>msmsply</td>
<td>202</td>
</tr>
<tr>
<td>mspct_classes</td>
<td>203</td>
</tr>
<tr>
<td>na.omit</td>
<td>204</td>
</tr>
<tr>
<td>normalization</td>
<td>206</td>
</tr>
<tr>
<td>normalize</td>
<td>207</td>
</tr>
<tr>
<td>normalized_diff_ind</td>
<td>211</td>
</tr>
<tr>
<td>normalize_range_arg</td>
<td>213</td>
</tr>
<tr>
<td>opaque.spct</td>
<td>214</td>
</tr>
<tr>
<td>oper_spectra</td>
<td>214</td>
</tr>
<tr>
<td>peaks</td>
<td>216</td>
</tr>
<tr>
<td>photodiode.spct</td>
<td>222</td>
</tr>
<tr>
<td>photons_energy_ratio</td>
<td>223</td>
</tr>
<tr>
<td>photon_irradiance</td>
<td>224</td>
</tr>
<tr>
<td>photon_ratio</td>
<td>226</td>
</tr>
<tr>
<td>plus-.generic_spct</td>
<td>227</td>
</tr>
<tr>
<td>polyester.spct</td>
<td>228</td>
</tr>
<tr>
<td>print</td>
<td>228</td>
</tr>
<tr>
<td>print.solar_time</td>
<td>229</td>
</tr>
<tr>
<td>print.summary_generic_spct</td>
<td>230</td>
</tr>
<tr>
<td>print.tod_time</td>
<td>231</td>
</tr>
</tbody>
</table>
R topics documented:

- print.waveband
- prod_spectra
- q2e
- qe_ratio
- q_fluence
- q_irrad
- q_ratio
- q_response
- rbindspct
- reflectance
- relative_AM
- replace_bad_pixs
- response
- Rfr_from_n
- rgb_spct
- rmDerivedMspct
- rmDerivedSpect
- round
- select_spct_attributes
- setBSWFUsed
- setFilterProperties
- setGenericSpect
- setHowMeasured
- setIdFactor
- setInstrDesc
- setInstrSettings
- setMultipleWl
- setNormalized
- setRfrType
- setScaled
- setTfrType
- setTimeUnit
- setWhatMeasured
- setWhenMeasured
- setWhereMeasured
- shared_member_class
- sign
- slash-generic_spct
- smooth_spct
- solar_time
- source_spct
- spct_attr2tb
- spct_classes
- spikes
- split2mspct
- split_bands
- split_energy_irradiance
- split_irradiance
R topics documented:

split_photon_irradiance ................................................. 301
spread ................................................................. 302
Subset ................................................................. 304
subset2mspec .......................................................... 305
subset_spectra .......................................................... 306
summary ............................................................... 307
summary_spect_classes ................................................ 308
sum_spectra ............................................................. 308
sun.daily.data ........................................................... 310
sun.daily.spect .......................................................... 311
sun.data ................................................................. 312
sun.spect ................................................................. 313
sun_angles ............................................................... 314
e_s_e_irrad2rgb ........................................................... 316
e_mean ................................................................. 317
e_mean_se ............................................................... 319
e_median ................................................................. 320
e_prod ................................................................. 321
e_range ................................................................. 323
e_sd ................................................................. 324
e_se ................................................................. 325
e_sum ................................................................. 327
e_var ................................................................. 328
T2A ................................................................. 330
T2Afr ................................................................. 331
tag ................................................................. 333
thin_wl ................................................................. 335
times_generic_spect ................................................ 338
transmittance ............................................................ 338
Trig ................................................................. 341
trimInstrDesc ............................................................ 342
trimInstrSettings .......................................................... 343
trim_spect ............................................................. 343
trim_tails .............................................................. 346
trim_waveband .......................................................... 347
trim_wl ............................................................... 349
tz_time_diff ............................................................ 351
uncollect ............................................................... 351
untag ................................................................. 353
upgrade_spect ............................................................ 354
upgrade_spectra .......................................................... 354
using_Tfr ............................................................... 355
validate_geocode .......................................................... 356
valleys ............................................................... 357
verbose_as_default ...................................................... 363
v_insert_hinges .......................................................... 363
v_replace_hinges ........................................................ 364
water_vp_sat ............................................................ 365
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. Astronomical calculations related solar angles and day length. 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 packages for analysis and plotting of data relevant to photobiology (described at https://www.r4photobiology.info/). The accompanying packages (under development) provide data and definitions that are to a large extent application-area specific while the functions in the present package are widely useful in photobiology and radiation quantification in geophysics and meteorology. Package 'photobiology' has its main focus in the characterization of the light environment in a biologically relevant manner and in the manipulation of spectral data to simulate photo-physical, photo-chemical and photo-biological interactions and responses. The focus of package 'pavo' (Maia et al., 2003) is in colour perception by animals and assessment of animal coloration. In spite of the different focus, there is some degree of overlap.
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.

Note

Code for some of the astronomical calculations has been adapted from that in package 'pavo'.

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


See Also

Useful links:

- https://docs.r4photobiology.info/photobiology
- https://bitbucket.org/aphalo/photobiology
- Report bugs at https://bitbucket.org/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)))
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 at 2856 K. CIE standard illuminant A is intended to represent typical, domestic, tungsten-filament lighting. Original data from [http://files.cie.co.at/204.xls](http://files.cie.co.at/204.xls) downloaded on 2014-07-25 The variables are as follows:

Usage

A.illuminant.spct

Format

A source spectrum with 96 rows and 2 variables

Details

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

Author(s)

CIE
See Also

Other Spectral data examples: D65.illuminant.spct, Ler_leaf.spct, Ler_leaf_rflt.spct, Ler_leaf_trns.spct, Ler_leaf_trns_i.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, green_leaf.spct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps_spct, white_led.raw_spct, white_led.source_spct, yellow_gel.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
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)

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

See Also

Other quantity conversion functions: Afr2T(), T2Afr(), T2A(), any2T(), as_quantum(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), 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,
## 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", "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".

- **wb.trim**
  - logical Flag indicating 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.

...  

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  

canonical 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)  

• default: Default for generic function  

• filter_spct: Specialization for filter spectra  

• object_spct: Specialization for object spectra  

• filter_mspct: Calculates absorbance from a filter_mspct  

• 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))
absorptance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "average")
absorptance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "total")
absorptance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "relative")
absorptance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "relative.pc")
absorptance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "contribution")
absorptance(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",
  ...
)

## 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",
  ...
)

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", "mean", "total", "contribution", "contribution.pc", "relative", "relative.pc".

wb.trim logical Flag 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.

... 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)

• default: Default for generic function
• filter_spct: Specialization for filter spectra
• object_spct: Specialization for object spectra
• filter_mspct: Calculates absorptance from a filter_mspct
• 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")
add_attr2tb  

Copy attributes from members of a generic_mspct

Description

Copy the when.measured, where.measured or what.measured attribute from members of a generic_mspct object into a tibble or data.frame.

Usage

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

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

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

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

Arguments

- **tb**: tibble or data.frame to which to add the data (optional).
- **mspct**: generic_mspct Any collection of spectra.
- **col.names**: named character vector Name(s) of column(s) to create. Values are the names of the attributes to copy, while if named, the names provide the name for the column.
idx character Name of the column with the names of the members of the collection of spectra.

Details

The attributes are copied to a column in a tibble or data frame. If the tb formal parameter receives NULL as argument, 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. If the argument to col.names is an named vector, with the names of members matching the names of attributes, then the values are used as names for the columns created. This permits setting any valid name for the new columns. If the vector passed to col.names has no names, then the values are interpreted as the names of the attributes to add, and also used as names for the new columns.

Value

A tibble With the metadata attributes in separate new variables.

Note

Currently supported attributes are "when.measured", "what.measured" and "where.measured". In the case of "where.measured", which has different components the name "where.measured" is ignored, but instead the following names are recognized: "lon" and "lat" for creating numeric columns of longitudes and latitudes respectively, and "geocode" for creating a column of data frames, in which case, if tb is not already a tibble it is converted into one before adding the new column. The order of the first two arguments is reversed in add_attr2tb() compared to the other functions. This is to allow its use in 'pipes', while the functions for single attributes are expected to be used mostly to create new tibbles.

Examples

```r
library(dplyr)

my.mspct <- source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))
q_irrad(my.mspct) %>%
  add_attr2tb(my.mspct, c(lat = "latitude",
     lon = "longitude",
     when.measured = "time"))

when_measured2tb(my.mspct)
```

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,
  ...
)

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

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 or 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)

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

See Also

Other quantity conversion functions: A2T(), T2Afr(), T2A(), any2T(), as_quantum(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), q2e()
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(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), 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 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 calibration_mspct object.
Methods (by class)

- default:
- data.frame:
- calibration_spct:
- list:
- 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.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

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

## Default S3 method:

```r
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)

- default:
as.chroma_mspct

See Also

 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.source_spct(), source_spct()

---

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)

- default:
- data.frame:
- chroma_spct:
- 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.source_mspct()`, `split2mspec()`, `subset2mspec()`

---

**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**

```r
as.chroma_spct(x, 
```

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

```r
## S4 generic method:
setGenericSpct
```

**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)**

- 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.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)
- default:
- data.frame:
- cps_spct:
- list:
- 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.source_mspct(), split2mspct(), subset2mspct()
as.filter_mspct

Value

A copy of x converted into a cps_spct object.

Methods (by class)

• 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.source_spct(), source_spct()

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(  
...  
)
as.filter_mspct

```r
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 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)**

- default:
- data.frame:
- filter_spct:
- list:
- 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`.
as.filter_spct

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.source_mspct(), split2mspct(), subset2mspct()

---

as.filter_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.filter_spct(x, ...)

## Default S3 method:

as.filter_spct(
  x,
  Tfr.type = c("total", "internal"),
  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"

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 filter_spct object.

Methods (by class)

• default:

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.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.
- **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 wavelength values 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)

- default:
- data.frame:
- generic_spct:
- list:
- 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.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)

• 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.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 length values can be 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,)
as.object_mspct

...)

## 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)

- default:
- data.frame:
- object_spct:
- 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.source_mspct(), split2mspct(), subset2mspct()
Coerce to a spectrum

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

Usage

```r
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)

- 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.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 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 \texttt{raw_mspct} object.

Methods (by class)
- default:
- data.frame:
- \texttt{raw_spct}:
- list:
- matrix:

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

See Also
Other Coercion methods for collections of spectra: \texttt{as.calibration_mspct()}, \texttt{as.chroma_mspct()}, \texttt{as.cps_mspct()}, \texttt{as.filter_mspct()}, \texttt{as.generic_mspct()}, \texttt{as.object_mspct()}, \texttt{as.reflector_mspct()}, \texttt{as.response_mspct()}, \texttt{as.source_mspct()}, \texttt{split2mspct()}, \texttt{subset2mspct()}

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

Usage
\[
\texttt{as.raw_spct}(x, \ldots)
\]

## Default S3 method:
\[
\texttt{as.raw_spct}(x, \ldots)
\]

Arguments
- \( x \) an R object
- \( \ldots \) other arguments passed to "set" functions
as.reflector_mspct

Description

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

Usage

```r
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,
  ...
)
```

Value

A copy of x converted into a raw_spct object.

Methods (by class)

- default:

See Also

setGenericSpct

Other constructs 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.source_spct()`, `source_spct()`
as.reflector_mspct

```r
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 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 reflector_mspct object.

**Methods (by class)**

- default:
- data.frame:
- reflector_spct:
- list:
- matrix:
as.reflector_spct

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 \( w.\text{length} \) vector must match one of the dimensions of \( x \).

See Also

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

---

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
as.reflector_spct(x, ...)  
## Default S3 method:  
as.reflector_spct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...  
)
\end{verbatim}

\textbf{Arguments}

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

\textbf{Value}

A copy of \( x \) converted into a \code{reflector_spct} object.

\textbf{Methods (by class)}

\begin{itemize}
  \item default:
\end{itemize}
See Also

setGenericSpct

Other constructors of spectral objects: \texttt{as.calibration_spct()}, \texttt{as.chroma_spct()}, \texttt{as.cps_spct()}, \texttt{as.filter_spct()}, \texttt{as.generic_spct()}, \texttt{as.object_spct()}, \texttt{as.raw_spct()}, \texttt{as.response_spct()}, \texttt{as.source_spct()}, \texttt{source_spct()}

Description

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

Usage

\begin{verbatim}
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_",
  ...
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
  \item \texttt{...} \hspace{1cm} passed to individual spectrum object constructor
  \item \texttt{time.unit} \hspace{1cm} character A string, "second", "day" or "exposure"
\end{itemize}
as.response_spct

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 response_mspct object.

Methods (by class)
- default:
  - data.frame:
  - response_spct:
  - list:
  - 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.source_mspct(), split2mspct(), subset2mspct()

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", ...)
as.solar_date

Arguments

x an R object

... other arguments passed to "set" functions
time.unit character A string, "second", "day" or "exposure"

Value

A copy of x converted into a response_spct object.

Methods (by class)

• 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.source_spct(), source_spct()

as.solar_date Convert a solar_time object into solar_date object

Description

Convert a solar_time object into solar_date object

Usage

as.solar_date(x, time)

Arguments

x solar_time object.
time an R date time object

Value

For method as.solar_date() a date-time object with the class attr set to "solar.time". This is needed only for unambiguous formatting and printing.
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**

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

**## Default S3 method:**

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

**## S3 method for class 'data.frame'**

```r
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'**

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

**## S3 method for class 'list'**

```r
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 'matrix'**

```r
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.

- **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 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 `source_mspct` object.

Methods (by class)

- default:
- data.frame:
- `source_spct`:
- list:
- 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()`, `split2mspct()`, `subset2mspct()`
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 A string, "second", "day" or "exposure"
bswf.used character
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)

• 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(), source_spct()
**as_energy**

*Convert spectral photon irradiance into spectral energy irradiance*

**Description**

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

**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**

\[
\text{with(sun.spct, as\_energy(w.length, s.q.irrad))}
\]

**as_quantum**

*Convert spectral energy irradiance into spectral photon irradiance*

**Description**

Convert spectral energy irradiance \([\text{W m}^{-2} \text{nm}^{-1}]\) into spectral photon irradiance expressed as number of photons \([\text{s}^{-1} \text{m}^{-2} \text{nm}^{-1}]\).

**Usage**

\[
as\_quantum(w\.length, s\.e\.irrad)
\]

**Examples**

\[
\text{with(sun.spct, as\_quantum(w.length, s.e.irrad))}
\]
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()`, `T2Afr()`, `T2A()`, `any2T()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`

Examples

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

Description

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

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

---

**as_tod**  
*Convert datetime to time-of-day*

**Description**

Convert a datetime into a time of day expressed in hours, minutes or seconds from midnight in local time for a time zone. This conversion is useful when time-series data for different days needs to be compared or plotted based on the local time-of-day.

**Usage**

```r
as_tod(x, unit.out = "hours", tz = NULL)
```

**Arguments**

- `x`  
a datetime object accepted by lubridate functions
- `unit.out`  
character string. One of "tod_time", "hours", "minutes", or "seconds".
- `tz`  
character string indicating time zone to be used in output.

**Value**

A numeric vector of the same length as `x`. If `unit.out = "tod_time"` an object of class "tod_time" which the same as for `unit.out = "hours"` but with the class attribute set, which dispatches to special `format()` and `print()` methods.

**Examples**

```r
library(lubridate)
my_instants <- ymd_hms("2020-05-17 12:05:03") + days(c(0, 30))
my_instants
as_tod(my_instants)
as_tod(my_instants, unit.out = "tod_time")
```
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**

```r
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**

```r
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**

```r
beesxyzCMF.spct
```

**Format**

A data frame with 81 rows and 4 variables
black_body.spct

Details

• w.length (nm)
• x
• y
• z

See Also


black_body.spct  Theoretical black body

Description

A dataset for a hypothetical object with transmittance 0/1 (0%), 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

**c**

*Combine collections of spectra*

**Description**

Combine two or more generic_mspct objects into a single object.

**Usage**

```r
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**

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

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

**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.
checkTimeUnit

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

See Also

Examples
ccd.spct
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: convertTfrType(), convertThickness(), convertTimeUnit(), getTimeUnit(), setRefUnit()

---

check_spct Check validity of spectral objects

Description

Check that an R object contains the expected data members.

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

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 '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, NA disables the test.
- **force**: logical If TRUE check is done even if checks are disabled.
- **multiple.wl**: numeric Maximum number of repeated w.length entries with same value.

### Methods (by class)

- **default**: Default for generic function.
- **generic_spct**: Specialization for generic_spct.
- **calibration_spct**: Specialization for calibration_spct.
- **raw_spct**: Specialization for raw_spct.
• cps_spct: Specialization for cps_spct.
• filter_spct: Specialization for filter_spct.
• reflector_spct: Specialization for reflector_spct.
• object_spct: Specialization for object_spct.
• response_spct: Specialization for response_spct.
• source_spct: Specialization for source_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

```r
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 used in calculations.

**Usage**

```r
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**

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

---

**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/](http://www.cvrl.org/) downloaded on 2014-04-29 The variables are as follows:

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

ciev10.spct

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also


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)
• x
• y
• z

Author(s)

CIE
**ciexyzCC10.spct**

**See Also**


**Examples**

`ciexyzCC10.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/](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

**Author(s)**

CIE

**See Also**


**Examples**

`ciexyzCC10.spct`
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

Author(s)

CIE

See Also


Examples

ciexyzCC2.spct
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/](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_spect object with 441 rows and 4 variables

Author(s)

CIE

See Also


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

Author(s)

CIE

See Also


Examples

ciexyzCMF2.spct
class_spct

Query which is the class of a spectrum

Description
Functions to check if an object is a generic spectrum, or coerce it if possible.

Usage
class_spct(x)

Arguments
x
any R object

Value
class_spct returns a vector containing all matching xxxx.spct 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 '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,
## 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 = NA_real_,
  range.s.data = c(NA_real_, NA_real_),
  fill = NA_real_,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
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)

- default: Default for generic function
- source_spct: Replace off-range values in a source spectrum
- filter_spct: Replace off-range values in a filter spectrum
- reflector_spct: Replace off-range values in a reflector spectrum
- object_spct: Replace off-range values in an object spectrum
- response_spct: Replace off-range values in a response spectrum
- cps_spct: Replace off-range values in a counts per second spectrum
- raw_spct: Replace off-range values in a raw counts spectrum
- generic_spct: Replace off-range values in a generic spectrum
### clear.spct

- source_mspct:
- filter_mspct:
- reflector_mspct:
- object_mspct:
- response_mspct:
- cps_mspct:
- raw_mspct:
- 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.

<table>
<thead>
<tr>
<th>clear.spct</th>
<th>Theoretical spectrum of a clear material</th>
</tr>
</thead>
</table>

**Description**

A dataset for a hypothetical object with transmittance 1/1 (100%)

**Usage**

clear.spct

**Format**

A filter_spct object with 4 rows and 2 variables

**Details**

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

**See Also**


**Examples**

clear.spct
clear_body.spct  
*Theoretical clear body*

**Description**

A dataset for a hypothetical object 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**


---

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**

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

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

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

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

## 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).

Value
A copy of x, most frequently of a shorter length, and never longer.

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

Note
The condition tested is \( \text{wl} \geq \text{range}[1] \& \text{wl} < (\text{range}[2] + 1e^{-13}) \).

See Also
Other trim functions: \text{trim_spct()}, \text{trim_waveband()}, \text{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))
**color_of**  
*Color of an object*

**Description**

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

**Usage**

`color_of(x, ...)`

```r
## 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)

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

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

```r
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)
```
Coarse-grained comparison of two spectra

Description

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

Usage

```r
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 wavebands, 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**

```r
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/](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.

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

**Description**

Function to set the "Tfr.type" attribute and simultaneously converting 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 uses as reference the light entering the object while total transmittance takes the incident light as reference. The conversion is possible only if reflectance is known. Either as spectral data in an object_spct object, or a filter_spct object that is under the hood an object_spct, or if a fixed reflectance factor applicable to all wavelengths is known.

Value

x possibly with the "thickness" field of the "filter.properties" attribute modified

Note

if x is not a filter_spct object, x is returned unchanged. If or x does not have the "filter.properties" attribute set and with no missing data, x is returned with Tfr set to NA values.

See Also

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

Examples

my.spct <- polyester.spct
filter_properties(my.spct) <- list(Rfr.constant = 0.07,
                                   thickness = 125e-6,
                                   attenuation.mode = "absorption")
convertTfrType(my.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 `convertTfrType()` 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

Note

if `x` is not a `filter_spct` object, `x` is returned unchanged. If or `x` does not have the "filter.properties" attribute set and with no missing data, `x` is returned with `Tfr` set to `NA` values.

See Also

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

Examples

```r
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)
### convolve_each

**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()`, `convertTfrType()`, `convertThickness()`, `getTimeUnit()`, `setTimeUnit()`

**Examples**

```r
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**

```r
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`, `^`, `div`, `log`, `minus`, `mod`, `plus`, `round`, `sign`, `slash`, `times`

---

**copy_attributes**

**Copy attributes**

**Description**

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

**Usage**

```r
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 default,
- `...` 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)

- default: Default for generic function
- generic_spct:
- generic_mspct:
- waveband:

---

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, ...)
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.
- ...: 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.
Description

A dataset containing fitted constants to be used as input for function D2_spectrum.

Format

A polynom::polynomial object with 6 constants.

Details

An object of class polynom::polynomial.

Author(s)

Lasse Ylianttila (data)
D2.UV654

Data for typical calibration lamps

Description

A dataset containing fitted constants to be used as input for function D2_spectrum.

Format

A polynom::polynomial object with 6 constants.

Details

An object of class polynom::polynomial.

Author(s)

Lasse Ylianttila (data)

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)

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 http://files.cie.co.at/204.xls downloaded on 2014-07-25 The variables are as follows:

Usage
D65.illuminant.spct

Format
A source spectrum with 107 rows and 2 variables

Details
• w.length (nm)
• s.e.irrad (rel. units)

Author(s)
CIE

See Also
Examples

D65.illuminant.spct

day_night

Times for sun positions

Description

Functions for calculating the timing of solar positions, given geographical coordinates and dates. They can be also used to find the time for an arbitrary solar elevation between 90 and -90 degrees by supplying "twilight" angle(s) as argument.

Usage

day_night(
  date = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "none",
  unit.out = "hours"
 )

day_night_fast(date, tz, geocode, twilight, unit.out)

noon_time(
  date = lubridate::today(),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "none",
  unit.out = "datetime"
 )

sunset_time(
  date = lubridate::today(),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "datetime"
 )

sunset_time(
  date = lubridate::today(),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "datetime"
day_length(
  date = lubridate::now(),
  tz = "UTC",
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "hours"
)

night_length(
  date = lubridate::now(),
  tz = "UTC",
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "hours"
)

Arguments

- **date**: "vector" of POSIXct times or Date objects, any valid TZ is allowed, default is current date at Greenwich.
- **tz**: character vector indicating time zone to be used in output.
- **geocode**: data frame with one or more rows and variables lon and lat as numeric values (degrees). If present, address will be copied to the output.
- **twilight**: character string, one of "none", "rim", "refraction", "sunlight", "civil", "nautical", "astronomical", or a numeric vector of length one, or two, giving solar elevation angle(s) in degrees (negative if below the horizon).
- **unit.out**: character string, One of "datetime", "day", "hour", "minute", or "second".

Details

Twilight names are interpreted as follows. "none": solar elevation = 0 degrees. "rim": upper rim of solar disk at the horizon or solar elevation = -0.53 / 2. "refraction": solar elevation = 0 degrees + refraction correction. "sunlight": upper rim of solar disk corrected for refraction, which is close to the value used by the online NOAA Solar Calculator. "civil": -6 degrees, "naval": -12 degrees, and "astronomical": -18 degrees. Unit names for output are as follows: "day", "hours", "minutes" and "seconds" times for sunrise and sunset are returned as times-of-day since midnight expressed in the chosen unit. "date" or "datetime" return the same times as datetime objects with TZ set (this is much slower than "hours"). Day length and night length are returned as numeric values expressed in hours when "datetime" is passed as argument to unit.out. If twilight is a numeric vector of length two, the element with index 1 is used for sunrise and that with index 2 for sunset.

Value

A tibble with variables day, tz, twilight.rise, twilight.set, longitude, latitude, address, sunrise, noon, sunset, daylength, nightlength or the corresponding individual vectors.

noon_time, sunrise_time and sunset_time return a vector of POSIXct times
day_night

day_length and night_length return numeric a vector giving the length in hours

Warning

Be aware that R's Date class does not save time zone metadata. This can lead to ambiguities in the current implementation based on time instants. The argument passed to date should be of class POSIXct, in other words an instant in time, from which the correct date will be computed based on the tz argument.

Note

This function is an implementation of Meeus equations as used in NOAA's on-line web calculator, which are very precise and valid for a very broad range of dates. For sunrise and sunset the times are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions. The more tangential to the horizon is the path of the sun, the larger the effect of refraction is on the times of visual occlusion of the sun behind the horizon—i.e. the largest timing errors occur at high latitudes. The computation is not defined for latitudes 90 and -90 degrees, i.e. at the poles.

There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function astrocalc4r in package 'fishmethods'. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the present implementation splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates.

In the current implementation functions sunrise_time, noon_time, sunset_time and day_length are wrappers on day_night, so if more than one quantity is needed it is preferable to directly call day_night as it will be faster.

night_length returns the length of night-time conditions in one day (00:00:00 to 23:59:59), rather than the length of the night between two consecutive days.

References


A different implementation is available at https://www.nefsc.noaa.gov/AstroCalc4R/ and in R package astrocalc4r. In 'fishmethods' (= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times.

An interactive web page using the same algorithms is available at https://www.esrl.noaa.gov/gmd/grad/solcalc/. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

sun_angles.

Other astronomy related functions: format.solar_time(), format.tod_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time(), sun_angles()
Examples

```r
library(lubridate)
my.geocode <- data.frame(lat = 60, lon = 25)
day_night(ymd("2015-05-30"), geocode = my.geocode)
day_night(ymd("2015-05-30") + days(1:10), geocode = my.geocode, twilight = "civil")
sunrise_time(ymd("2015-05-30"), geocode = my.geocode)
noon_time(ymd("2015-05-30"), geocode = my.geocode)
sunset_time(ymd("2015-05-30"), geocode = my.geocode)
day_length(ymd("2015-05-30"), geocode = my.geocode)
day_length(ymd("2015-05-30"), geocode = my.geocode, unit.out = "day")
```

---

**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(...)`

**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.
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,
## S3 method for class 'generic_spct'

```r
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)
```

## S3 method for class 'source_spct'

```r
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'

```r
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'

```r
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 '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,
  ...
)
.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,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
.
  .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,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
.
  .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.

### Value

`x` with rows corresponding to spikes replaced by a local average of adjacent neighbors outside the spike.

### Methods (by class)

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

```r
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

```r
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).
Arithmetic Operators

div-.generic_spct

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.lengthA numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.
- s.irradA 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(), 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)
Convert energy-based quantities into photon-based quantities.

Description

Function that converts spectral energy irradiance into spectral photon irradiance (molar).

Usage

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

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

## S3 method for class 'source_spct'
e2q(x, action = "add", 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
- 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.

Methods (by class)

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

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

```r
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()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`.

**Examples**

```r
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**

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

**See Also**

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

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()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2q()`, `q2e()`

Examples

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

**Description**

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

**Usage**

```r
enable_check_spct()

disable_check_spct()

set_check_spct(x)
```

Arguments

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

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

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**
```r
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**
```r
energy_irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
```
energy_ratio

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, by default as energy (W m-2 nm-1).
w.band waveband.
unit.in a character Allowed values "photon" or "energy", default is "energy".
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 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_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.
energy_ratio

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 (energy) irradiances (W m\(^{-2}\) nm\(^{-1}\)).

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 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

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

<table>
<thead>
<tr>
<th>eq_ratio</th>
<th>Energy:photon ratio</th>
</tr>
</thead>
</table>

Description

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

Usage

```r
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 = ifelse(naming != "none", ":[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 = ifelse(naming != "none", ":[e:q]", ":"),
    ...
)

attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
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.

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 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].
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

```r
## 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]
```

Methods (by class)

- default: Default for generic function
- source_spct: Method for source_spct objects
- source_mspct: Calculates energy: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_ratio()`, `q_ratio()`, `qe_ratio()`

Examples

```r
eq_ratio(sun.spct, new_waveband(400,700))
```
## 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 '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,
e_fluence

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,
  ...
)

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.

Methods (by class)

- default: Default for generic function
- source_spct: Calculate energy fluence from a source_spct object and the duration of the exposure.
- source_mspct: Calculates energy fluence 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 irradiance functions: e_irrad(), fluence(), irrad(), q_fluence(), q_irrad()
Examples

```r
library(lubridate)

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

---

**e_irrad**  
*Energy irradiance*

---

**Description**

Energy irradiance for one or more wavebands of a light source spectrum.

**Usage**

```r
e_irrad(  
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mulf,
  use.hinges,
  allow.scaled,
  ...
)
```

```
## Default S3 method:
e_irrad(  
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mulf,
  use.hinges,
  allow.scaled,
  ...
)
```

```
## S3 method for class 'source_spct'
e_irrad(  
  spct,
  w.band = NULL,
  quantity = "total",
```

```r
  ...
)
```
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",
...)

## 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",
  ...
)

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.

attr2tb character vector, see \texttt{add_attr2tb} for the syntax for \texttt{attr2tb} passed as is to formal parameter \texttt{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 \texttt{.export} and \texttt{.packages} arguments to supply them so that all cluster nodes have the correct environment set up for computing.

\textbf{Value}

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter \texttt{w.band}. A \texttt{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 \texttt{quantity} they can be re-expressed as relative fractions or percentages. In the case of vector output, \texttt{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 \texttt{time.unit} attribute is copied from the spectrum object to the output. Units are as follows: If units are absolute and \texttt{time.unit} is second, [W m\textsuperscript{-2} nm\textsuperscript{-1}] -> [W m\textsuperscript{-2}] If \texttt{time.unit} is day, [J d\textsuperscript{-1} m\textsuperscript{-2} nm\textsuperscript{-1}] -> [J m\textsuperscript{-2}]; if units are relative, fraction of one or percent.

\textbf{Methods (by class)}

- \texttt{default}: Default for generic function
- \texttt{source_spct}: Calculates energy irradiance from a \texttt{source_spct} object.
- \texttt{source_mspct}: Calculates energy irradiance from a \texttt{source_mspct} object.

\textbf{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 \texttt{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 \texttt{w.length} vector.

\textbf{See Also}

Other irradiance functions: \texttt{e_fluence()}, \texttt{fluence()}, \texttt{irrad()}, \texttt{q_fluence()}, \texttt{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")

Description

This function returns the photon ratio for a given pair of wavebands of a light source spectrum.

Usage

```r
 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.mutl = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[e:e]", ""),
  ...)

## 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.mutl = FALSE,
  use.hinges = NULL,
  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) 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.mutl**
  - 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.

Value

In the case of methods for individual spectra, a numeric vector of adimensional values giving a energy ratio between integrated energy irradiances for pairs of wavebands, with name attribute set to the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used, with "(e:e)" appended. A data.frame in the case of collections of spectra, containing one column for each ratio 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.

Ratio definitions are "assembled" from the arguments passed to w.band.num and w.band.denom. If both arguments are of equal length, then the wavebands are paired to obtain as many ratios 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)

- default: Default for generic function
- source_spect: Method for source_spect objects
- source_mspct: Calculates energy:energy ratio from a source_mspct object.

Note

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.
e_response

See Also
Other photon and energy ratio functions: `eq_ratio()`, `q_ratio()`, `qe_ratio()`

Examples
```r
e_ratio(sun.spct, new_waveband(400,500), new_waveband(400,700))
```

---

### 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**

```r
e_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)
```

```r
## Default S3 method:
e_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)
```

```r
## 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.

**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)
- default: Default method for generic function
- 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()
Description

A dataset containing fitted constants to be used as input for function FEL_spectrum.

Format

A numeric vector.

Author(s)

Lasse Ylianttila (data)

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 function is valid for wavelengths in the range 250 nm to 900 nm, for wavelengths outside this range NAs are returned.
findMultipleWl

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 peaks in a spectrum

Description

This function finds all peaks (local maxima) in a spectrum, using a user provided size threshold relative to the tallest peak (global maximum) bellow which found peaks are ignored—i.e., not included in the returned value. This is a wrapper built on top of function peaks() from package ‘splus2R’.

Usage

find_peaks(x, ignore_threshold = 0, span = 3, strict = TRUE, na.rm = FALSE)
**find_spikes**

**Arguments**

- **x**: numeric vector
- **ignore_threshold**: numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.
- **span**: integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use NULL for the global peak.
- **strict**: logical 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.

**Value**

A logical vector of the same length as x. Values that are TRUE correspond to local peaks in the data.

**Note**

This function is a wrapper built on function peaks from splus2R and handles non-finite (including NA) values differently than splus2R::peaks, instead of giving an error they are replaced with the smallest finite value in x.

**See Also**

peaks

Other peaks and valleys functions: find_spikes(), get_peaks(), peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

**Examples**

```r
with(sun.data, w.length[find_peaks(s.e.irrad)])
```

**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 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. Other kinds of accidental "outlayers" will be also detected.
Usage

```r
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


See Also

Other peaks and valleys functions: `find_peaks()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `valleys()`, `wls_at_target()`

Examples

```r
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**

```r
find_wls(
  x,
  target = NULL,
  col.name.x = NULL,
  col.name = NULL,
  .fun = `<=`,
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE
)
```

**Arguments**

- **x**: an R object
- **target**: numeric value indicating the spectral quantity value for which wavelengths are to be searched and interpolated if need. The character strings "half.maximum" and "half.range" are also accepted as arguments.
- **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_spect" 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 straddling the target.
- **idfactor**: logical or character Generates an index column of factor type. If idfactor = TRUE then the column is auto named spct.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 wavelengths for the target value is returned.
**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**

```r
fit_peaks(
  x,
  peaks.idx,
  span,
  x.col.name = NULL,
  y.col.name,
  method,
  max.span = 5L,
  maximum = TRUE,
  keep.cols = NULL
)
```

**Note**

This function is used internally by method `wls_at_target()`, and these methods should be preferred in user code and scripts.

**Examples**

```r
find_wls(white_led.source_spct)
find_wls(white_led.source_spct, target = "half.maximum")
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("HM", "HR"), idfactor = TRUE)
find_wls(white_led.source_spct, target = c("HM", "HR"), interpolate = TRUE)

led.df <- as.data.frame(white_led.source_spct)
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_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 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

Value

An R object of the same class as x containing the fitted values for the peaks, and optionally the values for at 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 "quanta-

exposure.time lubridate::duration object.

scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier

wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if

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

allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are

... 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.
fluence

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]

Methods (by class)

- default: Default for generic function
- source_spct: Calculate photon fluence from a source_spct object and the duration of the exposure
- source_mspct: Calculates fluence 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 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") )
Description
Format a solar_time object for pretty printing

Usage
## S3 method for class 'solar_time'
format(x, ..., sep = ":")

Arguments
x an R object
... ignored
sep character used as separator

See Also
Other astronomy related functions: day_night(), format.tod_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time(), sun_angles()

Description
Format a tod_time object for pretty printing

Usage
## S3 method for class 'tod_time'
format(x, ..., sep = ":")

Arguments
x an R object
... ignored
sep character used as separator

See Also
Other astronomy related functions: day_night(), format.solar_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time(), sun_angles()
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 functions return a spectral object of the same class as the one supplied as argument but with the spectral data rescaled based on summary function f applied over a specific range or wavelengths and a target value for the summary value.

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 '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,
  ...
)

## S3 method for class 'raw_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...
)

## S3 method for class 'cps_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...
)

## S3 method for class 'generic_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  col.names,
  ...,
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.

Value

A copy of 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)

- default: Default for generic function
- source_spct:
- response_spct:
- filter_spct:
- reflector_spct:
- raw_spct:
- cps_spct:
- generic_spct:
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.

Note

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, this package uses an attribute, which will by default be set assuming the first of these two situations when target == 1 and the second situation otherwise. These defaults can be overriden with an explicit logical argument passed to set.scaled.

See Also

Other rescaling functions: fshift(), getNormalized(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

Examples

```r
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 <- fshift(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))
```
Usage

fshift(x, ...)

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

## S3 method for class 'source_spct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
fshift(x, range = c(min(x), min(x) + 10), f = "min", qty.out = NULL, ...)

## S3 method for class 'source_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'raw_spct'
fshift(x, range = c(min(x), min(x) + 10), f = "mean", qty.out = NULL, ...)
## S3 method for class 'cps_spct'
fshift(x, range = c(min(x), min(x) + 10), f = "mean", qty.out = NULL, ...)

## S3 method for class 'generic_spct'
fshift(x, range = c(min(x), min(x) + 10), f = "mean", col.names, ...)

## S3 method for class 'response_mspct'
fshift(
  x,
  range = c(min(x), 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(min(x), 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(min(x), min(x) + 10),
  f = "min",
  qty.out = NULL,
  ..., .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  ..., .parallel = FALSE,
  .paropts = NULL
)
## S3 method for class 'cps_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  ..., 
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
fshift(
  x,
  range = c(min(x), 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)

- default: Default for generic function
- source_spct:
- response_spct:
- filter_spct:
- reflector_spct:
- source_mspct:
- raw_spct:
- cps_spct:
- generic_spct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- raw_mspct:
- cps_mspct:
- generic_mspct:

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-spect 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, ...)
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.
• 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))
getBSWFUsed  
*Get the "bswf.used" attribute*

**Description**

Function to read the "time.unit" attribute of an existing source_spct object

**Usage**

```r
getBSWFUsed(x)
```

**Arguments**

- `x`  
a source_spct object

**Value**

character string

**Note**

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

**See Also**

Other BSWF attribute functions: `setBSWFUsed()`

**Examples**

```r
getBSWFUsed(sun.spct)
```

---

getFilterProperties  
*Get the "filter.properties" attribute*

**Description**

Function to read the "filter.properties" attribute of an existing filter_spct or a filter_mspct.
getFilterProperties

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, ..., 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", "thickness" 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)

• default: default
• filter_spct: generic_spct
• summary_filter_spct: summary_generic_spct
• 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: getHowMeasured(), getInstrDesc(), getInstrSettings(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(),
getHowMeasured

```r
```

**Examples**

```r
filter_properties(polyester.spct)
```

---

**getHowMeasured**

*Get the "how.measured" attribute*

**Description**

Function to read the "how.measured" attribute of an existing generic_spct or a generic_mspct.

**Usage**

```r
getHowMeasured(x, ...)
```

## Default S3 method:
```
getHowMeasured(x, ...)
```

## S3 method for class 'generic_spct'
```
getHowMeasured(x, ...)
```

## S3 method for class 'summary_generic_spct'
```
getHowMeasured(x, ...)
```

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

**Arguments**

- `x` a generic_spct object
- `...` 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**

character vector An object containing a description of the data.
Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

Note

The method for collections of spectra returns the a tibble with a column of character strings.

See Also


Examples

how_measured(sun.spct)
getInstrDesc

See Also

Other idfactor attribute functions: setIdFactor()

Examples

getMultipleWl(sun.spct)

---

**getInstrDesc**  
Get the "instr.desc" attribute

Description

Function to read the "instr.desc" attribute of an existing generic_spct object.

Usage

getInstrDesc(x)

Arguments

x  
a generic_spct object

Value

list (depends on instrument type)

See Also

getInstrSettings

Get the "instr.settings" attribute

Description

Function to read the "instr.settings" attribute of an existing generic_spct object.

Usage

getInstrSettings(x)

Arguments

x a generic_spct object

Value

list

See Also

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

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_spct 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 read the "multiple.wl" attribute of an existing generic_spct.

Usage

getMultipleWl(x)

Arguments

x a generic_spct object

Value

integer

Note

If x is not a generic_spct or an object of a derived class NA is returned.

See Also

Other multiple.wl attribute functions: setMultipleWl()

Examples

getMultipleWl(sun.spct)
**getNormalized**

Get the "normalized" attribute

**Description**

Function to read the "normalized" attribute of an existing generic_spct object.

**Usage**

getNormalized(x)

getNormalised(x)

**Arguments**

x  
a generic_spct object

**Value**

character or numeric or logical

**Note**

if x is not a generic_spct object, NA is returned

getNormalised() is a synonym for this getNormalized() method.

**See Also**

Other rescaling functions: fscale(), fshift(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

---

**getRfrType**

Get the "Rfr.type" attribute

**Description**

Function to read the "Rfr.type" attribute of an existing reflector_spct object or object_spct object.

**Usage**

getRfrType(x)

**Arguments**

x  
a source_spct object
getScaled

Value

character string

Note

if x is not a filter_spct object, NA is returned

See Also

Other Rfr attribute functions: setRfrType()

---

getscaled  Get the "scaled" attribute

Description

Function to read the "scaled" attribute of an existing generic_spct object.

Usage

getScaled(x)

Arguments

x  a generic_spct object

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)
**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.

---

**getTfrType**  
*Get the "Tfr.type" attribute*

---

**Description**  
Function to read the "Tfr.type" attribute of an existing filter_spct or object_spct object.

**Usage**  
getTfrType(x)

**Arguments**  
- x  
  a filter_spct or object_spct object

**Value**  
character string

**Note**  
If x is not a filter_spct or an object_spct object, NA is returned.
See Also

Other Tfr attribute functions: `setTfrType()`

Examples

```r
getTfrType(polyester.spct)
```

---

### `getTimeUnit`

*Get the "time.unit" attribute of an existing source_spct object*

**Description**

Function to read the "time.unit" attribute.

**Usage**

```r
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()`, `convertTfrType()`, `convertThickness()`, `convertTimeUnit()`, `setTimeUnit()`

**Examples**

```r
getTimeUnit(sun.spct)
```
getWhatMeasured

Get the "what.measured" attribute

Description

Function to read the "what.measured" attribute of an existing generic_spct or a generic_mspct.

Usage

getWhatMeasured(x, ...)
what.measured(x, ...)

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

## S3 method for class 'generic_spct'
getWhatMeasured(x, ...)

## S3 method for class 'summary_generic_spct'
getWhatMeasured(x, ...)

## S3 method for class 'generic_mspct'
getWhatMeasured(x, ..., idx = "spct.idx")

Arguments

x a generic_spct object
...
 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.

Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

Note

The method for collections of spectra returns the a tibble with a column of character strings.
getWhenMeasured

See Also

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

Examples

what_measured(sun.spct)

getWhenMeasured

Get the "when.measured" attribute

Description

Function to read the "when.measured" attribute of an existing generic_spct or a generic_mspct.

Usage

getWhenMeasured(x, ...)

when_measured(x, ...)

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

## S3 method for class 'generic_spct'
getWhenMeasured(x, ...)

## S3 method for class 'summary_generic_spct'
getWhenMeasured(x, ...)

## S3 method for class 'generic_mspct'
getWhenMeasured(x, ..., idx = "spct.idx")

Arguments

x a generic_spct object

... 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.
getWhereMeasured

Value

POSIXct An object with date and time.

Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

Note

If x is not a generic_spct or an object of a derived class NA is returned.

The method for collections of spectra returns the a tibble with the correct times in TZ = "UTC".

See Also


Examples

when_measured(sun.spct)

getWhereMeasured

Get the "where.measured" attribute

Description

Function to read the "where.measured" attribute of an existing generic_spct.

Usage

gewhereMeasured(x, ...)

where_measured(x, ...)

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

## S3 method for class 'generic_spct'
getWhereMeasured(x, ...)

## S3 method for class 'summary_generic_spct'
getWhereMeasured(x, ...)

## S3 method for class 'generic_mspct'
getWhereMeasured(x, ..., idx = "spct.idx")

### Arguments

- **x**: a generic_spct object
- **...**: 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 data.frame with a single row and at least columns "lon" and "lat".

### Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

### Note

If x is not a generic_spct or an object of a derived class NA is returned.

### See Also


### Examples

where_measured(sun.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 '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.
get_peaks

Value

Named list of attribute values.

Methods (by class)

• generic_spct: generic_spct
• source_spct: source_spct
• filter_spct: filter_spct
• reflector_spct: reflector_spct
• object_spct: object_spct
• waveband: waveband

See Also

select_spct_attributes

Other measurement metadata functions: getFilterProperties(), getHowMeasured(), getInstrDesc(),
getInstrSettings(), getWhatMeasured(), getWhereMeasured(), isValidInstrDesc(),
isValidInstrSettings(), select_spct_attributes(), setFilterProperties(), setHowMeasured(),
setInstrDesc(), setInstrSettings(), setWhatMeasured(), setWhereMeasured(),
spct_attr2tb(), trimInstrDesc(), trimInstrSettings()

description

Get peaks and valleys in a spectrum

Description

These functions find peaks (local maxima) or valleys (local minima) in a spectrum, using a user
selectable size threshold relative to the tallest peak (global maximum). This a wrapper built on top
of function peaks from package spls2R.

Usage

get_peaks(
  x,
  y,
  ignore_threshold = 0,
  span = 5,
  strict = TRUE,
  x_unit = "",
  x_digits = 3,
  na.rm = FALSE
)

get_valleys(
  x,
get_peaks

y,  
ignore_threshold = 0,  
span = 5,  
strict = TRUE,  
x_unit = "",  
x_digits = 3,  
na.rm = FALSE)

Arguments

x numeric
y numeric
ignore_threshold numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.
span integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use NULL for the global peak.
strict logical 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.

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.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples

with(sun.spct, get_peaks(w.length, s.e.irrad))
with(sun.spct, get_valleys(w.length, s.e.irrad))
green_leaf.spct

---

**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, D65.illuminant.spct, Ler_leaf.spct, Ler_leaf_rfl1.spct, Ler_leaf_trns.spct, Ler_leaf_trns_i.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps_spct, white_led.raw_spct, white_led.source_spct, yellow_gel.spct*

**Examples**

```
green_leaf.spct
```
head_tail

Return the First and Last Part 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)

- default:
- data.frame:
- matrix:
- function:
- table:
- 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'. I have been missing this method especially 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(letters)
head_tail(letters, n = -6L)
head_tail(freeny.x, n = 10L)
head_tail(freeny.y)

head_tail(stats::ftable(Titanic))

---

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)
**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

```r
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.

Examples

```r
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\(^{-1}\)] \(\rightarrow\) [W m\(^{-2}\)]

See Also

Other low-level functions operating on numeric vectors: \texttt{as\_energy()}, \texttt{as\_quantum\_mol()}, \texttt{calc\_multipliers()}, \texttt{div\_spectra()}, \texttt{energy\_irradiance()}, \texttt{energy\_ratio()}, \texttt{insert\_hinges()}, \texttt{interpolate\_spectrum()}, \texttt{irradiance()}, \texttt{l\_insert\_hinges()}, \texttt{oper\_spectra()}, \texttt{photon\_irradiance()}, \texttt{photon\_ratio()}, \texttt{photons\_energy\_ratio()}, \texttt{prod\_spectra()}, \texttt{s\_e\_irrad2rgb()}, \texttt{split\_energy\_irradiance()}, \texttt{split\_photon\_irradiance()}, \texttt{subt\_spectra()}, \texttt{sum\_spectra()}, \texttt{trim\_tails()}, \texttt{v\_insert\_hinges()}, \texttt{v\_replace\_hinges()}

Examples

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

interpolate_spect

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

If length.out it is a numeric value, then gives the number of rows in the output, if it is NULL, the values in the numeric vector w.length.out are used. If both are not NULL then the range of w.length.out and length.out are used to generate a vector of wavelength. A value of NULL for fill prevents extrapolation. If both w.length.out and length.out are NULL the input is returned as is. If w.length.out has length equal to zero, zero rows from the input are returned.

Value

A new spectral object of the same class as argument spct.

Note

The default fill = NA fills extrapolated values with NA. Giving NULL as argument for fill deletes wavelengths outside the input data range from the returned spectrum. A numerical value can be also be provided as fill. This function calls interpolate_spectrum for each non-wavelength column in the input spectra object.
interpolate_spectrum

Examples

interpolate_spect(sun.spct, 400:500, NA)
interpolate_spect(sun.spct, 400:500, NULL)
interpolate_spect(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_spect(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().

Value

a numeric vector of interpolated spectral values.

Note

The current version of interpolate uses spline if fewer than 25 data points are available. Otherwise it uses approx. In the first case a cubic spline is used, in the second case linear interpolation, which should be faster.

See Also

splinefun.

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

\[
\text{my.w.length} \leftarrow 300:700 \\
\text{with(sun.data, interpolate_spectrum(w.length, s.e.irrad, my.w.length))}
\]

---

**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

\[
\text{interpolate_wl}(x, \text{w.length.out}, \text{fill}, \text{length.out}, \ldots)
\]

- **Default S3 method:**
  \[
  \text{interpolate_wl}(x, \text{w.length.out}, \text{fill}, \text{length.out}, \ldots)
  \]

- **S3 method for class 'generic_spct'**
  \[
  \text{interpolate_wl}(x, \text{w.length.out} = \text{NULL}, \text{fill} = \text{NA}, \text{length.out} = \text{NULL}, \ldots)
  \]

- **S3 method for class 'generic_mspct'**
  \[
  \text{interpolate_wl}(x, \text{w.length.out} = \text{NULL}, \text{fill} = \text{NA}, \text{length.out} = \text{NULL}, \ldots, \text{.parallel} = \text{FALSE}, \text{.paropts} = \text{NULL})
  \]

### 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

- **.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.
irrad

Details

If `length.out` it is a numeric value, then gives the number of rows in the output, if it is `NULL`, the values in the numeric vector `w.length.out` are used. If both are not `NULL` then the range of `w.length.out` and `length.out` are used to generate a vector of wavelength. A value of `NULL` for `fill` prevents extrapolation.

Value

A new spectral object of the same class as argument `spct`.

Methods (by class)

- default: Default for generic function
- `generic_spct`: Interpolate wavelength in an object of class "generic_spct" or derived.
- `generic_mspct`: Interpolate wavelength in an object of class "generic_mspct" or derived.

Note

The default `fill = NA` fills extrapolated values with NA. Giving `NULL` as argument for `fill` deletes wavelengths outside the input data range from the returned spectrum. A numerical value can be also be provided as fill. This function calls `interpolate_spectrum` for each non-wavelength column in the input spectra object.

Examples

```r
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

```r
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 = getOption("photobiology.use.hinges"),
allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...)

## 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 string with 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.
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 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. 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]

Methods (by class)

- `default`: Default for generic function
- `source_spct`: Calculates irradiance from a `source_spct` object.
- `source_mspct`: Calculates irradiance 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.

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 wave-lengths 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 character 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 or a vector of numeric values with no change in scale factor: [W m-2 nm-1] -> [mol s-1 m-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

```r
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**

```r
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.generic_spct

is.reflector_mspct(x)

is.object_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)

Description

Functions to check if an object is of a given type of spectrum, or coerce it if possible.

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.old_spct

is.filter_spct(x)

is.reflector_spct(x)

is.object_spct(x)

is.chroma_spct(x)

is.any_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**

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 as used in photobiology (>= 0.6.0).

**Usage**

is.old_spct(object)
is.solar_time

Arguments

object an R object

Value

logical

See Also

Other upgrade from earlier versions: upgrade_spct(), upgrade_spectra()

is.solar_time

Query class

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_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

```r
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.
isValidInstrDesc

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.

Usage

isValidInstrDesc(x)

Arguments

x a generic_spct object

Value

logical TRUE if at least instrument name and serial number is found.

See Also

is_ValidInstrSettings  

Check the "instr.settings" attribute

Description

Function to validate the "instr.settings" attribute of an existing generic_spct object.

Usage

isValidInstrSettings(x)

Arguments

x  
a generic_spct object

Value

logical TRUE if at least integration time data is found.

See Also


is_absorbance_based  

Query if a spectrum contains absorbance or transmittance data

Description

Functions to check 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
is_effective

Value

is_absorbance_based returns TRUE if its argument is a filter_spct object that contains spectral absorbance 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.

is_absorptance_based returns TRUE if its argument is a filter_spct object that contains spectral absorbance 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.

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_photon_based()

Examples

is_absorbance_based(polyester.spct)
my.spect <- T2A(polyester.spct)
is.filter_spct(my.spect)
is.absorbance_based(my.spect)

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)
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.

```r
## 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)

- default: Default method.
- waveband: Is a waveband object defining a method for calculating effective irradiance.
- generic_spct: Does a source_spct object contain effective spectral irradiance values.
- source_spct: Does a source_spct object contain effective spectral irradiance values.
- summary.generic_spct: Method for "summary_generic_spct".
- summary_source_spct: Method for "summary_source_spct".

See Also

Other waveband attributes: labels(), normalization()

Examples

```r
is_effective(summary(sun.spct))
```
is\_photon\_based

Usage

is\_normalized(x)

is\_normalised(x)

Arguments

x

An R object.

Value

A logical value. If x is not normalized or x is not a generic\_spct object the value returned is FALSE.

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 check if source\_spct and response\_spct objects contains 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 TRUE if its argument is a source\_spct or a response\_spct object that contains photon base data and FALSE if such an object does not contain such data, but returns NA for any other R object, including those belonging other generic\_spct-derived classes.

is\_energy\_based returns TRUE if its argument is a source\_spct or a response\_spct object that contains energy base data and FALSE if such an object does not contain such data, but returns NA for any other R object, including those belonging other generic\_spct-derived classes.
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 a spct object contains tags.

Usage

is_tagged(x)

Arguments

x  any R object

Value

is_tagged returns TRUE if its argument is 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 homogenous 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, filling the spectral data for missing wave lengths in individual spectra with NA.
join_mspct

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, ...)

## S3 method for class 'source_mspct'
join_mspct(x, type = "full", unit.out = "energy", ...)

## S3 method for class 'response_mspct'
join_mspct(x, type = "full", unit.out = "energy", ...)

## S3 method for class 'filter_mspct'
join_mspct(x, type = "full", qty.out = "transmittance", ...)

## S3 method for class 'reflector_mspct'
join_mspct(x, type = "full", ...)

## S3 method for class 'object_mspct'
join_mspct(x, type = "full", qty.out, ...)

Arguments

x A generic_mspct object, or an object of a class derived from generic_mspct.
type character Type of join: "left", "right", "inner" or "full" (default). 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".
unit.out character Allowed values "energy", and "photon", or its alias "quantum".
qty.out character Allowed values "transmittance", and "absorbance".

Value

An object of class dataframe, with the spectra joined by wave length, with rows in addition sorted by wave length (variable w.length).

Methods (by class)

- default:
- generic_mspct:
- source_mspct:
- response_mspct:
labels

- filter_mspct:
- reflector_mspct:
- object_mspct:

Note

Currently only generic_spct, source_mspct, response_mspct, filter_mspct, reflector_mspct and object_mspct classes have this method implemented.

Find labels from "waveband" object

Description

A function to obtain 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)

- 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

### Format
An object_spct object with 2401 rows and 3 variables

### 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, D65.illuminant.spct, Ler_leaf_rflt.spct, Ler_leaf_trns.spct, Ler_leaf_trns_i.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, green_leaf.spct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps_spct, white_led.raw_spct, white_led.source_spct, yellow_gel.spct*

### Examples
Ler_leaf.spct
**Ler_leaf_rflt.spct**

---

**Ler_leaf_rflt.spct  Green Arabidopsis leaf spectral reflectance.**

---

**Description**

A dataset of total spectral reflectance expressed as fractions of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf_rflt.spct

**Format**

An reflector_spct object with 1750 rows and 2 variables

**Details**

- w.length (nm)
- Rfr (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**


**Examples**

Ler_leaf_rflt.spct
Ler_leaf_trns.spct

---

**Ler_leaf_trns.spct**  
*Green Arabidopsis leaf spectral transmittance.*

---

**Description**

A dataset of total spectral transmittance expressed as a fraction of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf_trns.spct

**Format**

An `filter_spct` object with 1753 rows and 2 variables

**Details**

- `w.length (nm)`
- `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**


**Examples**

Ler_leaf_trns.spct
**Ler_leaf_trns_i.spct**  

*Green Arabidopsis leaf spectral transmittance.*

**Description**

A dataset of internal spectral transmittance expressed as a fraction of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf_trns_i.spct

**Format**

An `filter_spct` object with 2401 rows and 2 variables

**Details**

- w.length (nm)
- 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, D65.illuminant.spct, Ler_leaf.spct, Ler_leaf_rflt.spct, Ler_leaf_trns.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, green_leaf.spct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps_spct, white_led.raw_spct, white_led.source_spct, yellow_gel.spct*

**Examples**

Ler_leaf_trns_i.spct
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

```r
## S3 method for class 'generic_spct'
log(x, base = exp(1))
```

```r
log2.generic_spct(x)
log10.generic_spct(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, \(^{\text{generic_spct}}\), convolve_each(), div\text{-}\text{generic_spct}, minus\text{-}\text{generic_spct}, mod\text{-}\text{generic_spct}, plus\text{-}\text{generic_spct}, round(), sign(), slash\text{-}\text{generic_spct}, times\text{-}\text{generic_spct}
**MathFun**  

**Miscellaneous Mathematical Functions**

**Description**

\texttt{abs(x)} computes the absolute value of \( x \), \texttt{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**

\begin{verbatim}
## S3 method for class 'generic_spct'
sqrt(x)

## S3 method for class 'generic_spct'
abs(x)
\end{verbatim}

**Arguments**

\( x \) an object of class "generic_spct"

**See Also**

Other math operators and functions: \texttt{^\.generic_spct()}, \texttt{convolve_each()}, \texttt{div-.generic_spct}, \texttt{log()}, \texttt{minus-.generic_spct}, \texttt{mod-.generic_spct}, \texttt{plus-.generic_spct}, \texttt{round()}, \texttt{sign()}, \texttt{slash-.generic_spct}, \texttt{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**

\begin{verbatim}
merge2object_spct(
  x,
  y,
  by = "w.length",
  ...,
  w.length.out = x[["w.length"]],
  Tfr.type.out = "total"
)
\end{verbatim}
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

```r
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,
  ...)
```
minus-.generic_spct

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)

- default: Default for generic function
- generic_spct:

---

### minus-.generic_spct

**Arithmetic Operators**

Description

Subtraction operator for generic spectra.

Usage

```r
## 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

```r
## 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-spect transform methods*

Description

Apply a function or operator to a collection of spectra.

Usage

```r
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
a data frame in the case of msdply
a list in the case of mslply
an vector in the case of msaply

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()
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**

```r
## 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 '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, ...)
```
na.omit

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 '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 can 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 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)

Arguments

x an R object
Methods (by class)

- default: Default methods.
- waveband: Normalization of a waveband object.

See Also

Other waveband attributes: is_effective(), labels()

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.

Usage

normalize(x, ...)

normalise(x, ...)

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

## S3 method for class 'source_spct'
normalize(
  x,
  ...,  
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE
)

## S3 method for class 'response_spct'
normalize(
  x,
  ...,  
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE
)

## S3 method for class 'filter_spct'


normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  na.rm = FALSE
)

## S3 method for class 'reflector_spct'
normalize(x, ..., range = NULL, norm = "max", qty.out = NULL, na.rm = FALSE)

## S3 method for class 'raw_spct'
normalize(x, ..., range = NULL, norm = "max", na.rm = FALSE)

## S3 method for class 'cps_spct'
normalize(x, ..., range = NULL, norm = "max", na.rm = FALSE)

## S3 method for class 'generic_spct'
normalize(x, ..., range = NULL, norm = "max", col.names, na.rm = FALSE)

## S3 method for class 'source_mspct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
normalize(
  x,
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.

**unit.out**
character Allowed values "energy", and "photon", or its alias "quantum"

**na.rm**
logical indicating whether \( \text{NA} \) values should be stripped before calculating the summary (e.g. "max") used for normalization.

**qty.out**
character string Allowed values are "transmittance", and "absorbance" indicating on which quantity to apply the normalization.

**col.names**
character vector containing the names of columns or variables to which to apply the normalization.

**.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 a wavelength retrieved using an arbitrary R function applied to the spectrum. By default the function is applied to the whole spectrum, but by passing a range of wavelengths as input, the search can be limited to a region of interest within the spectrum.

### Value
A copy of \( x \), with spectral data values normalized to one for the criterion specified by the argument passed to **norm**.

A copy of \( x \) 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 \( x \).

### Methods (by class)
- **default**: Default for generic function
- **source_spct**: Normalize a source_spct object.
- **response_spct**: Normalize a response spectrum.
- **filter_spct**: Normalize a filter spectrum.
- **reflector_spct**: Normalize a reflector spectrum.
- **raw_spct**: Normalize a raw spectrum.
- **cps_spct**: Normalize a cps spectrum.
- **generic_spct**: Normalize a raw spectrum.
- **source_mspct**: Normalize the members of a source_mspct object.
• response_mspct: Normalize the members of a response_mspct object.
• filter_mspct: Normalize the members of a filter_mspct object.
• reflector_mspct: Normalize the members of a reflector_mspct object.
• raw_mspct: Normalize the members of a raw_mspct object.
• cps_mspct: Normalize the members of a cps_mspct object.

Note

normalise() is a synonym for this normalize() method.

1) 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 stripping NA values, and only the values that were NA in x will be NA values in the returned spectrum.

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 index.

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 them is in the wavebands used.

Usage

normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
normalised_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
NDxI(spct, plus.w.band, minus.w.band, f, ...)
### Default S3 method:

```r
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
```

### S3 method for class 'generic_spct'

```r
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
```

### S3 method for class 'generic_mspct'

```r
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
```

#### Arguments

- **spct**: an R object
- **plus.w.band**: waveband objects. The waveband determine the region of the spectrum used in the calculations.
- **minus.w.band**: waveband objects. The waveband determine the region 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. Wavebands can describe weighting functions if desired.

#### 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)

- default: default
- generic_spct:
- 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 reflectance as input, waveband definitions provided by package `photobiologyWavebands` can be used.

`normalised_diff_ind()` is a synonym for `normalized_diff_ind()`.

`NDxI()` is a shorthand for `normalized_diff_ind()`.
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**

```r
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**

```r
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)
```
opaque.spct  
*Theoretical spectrum of an opaque material*

**Description**

A dataset for a hypothetical object with transmittance 0/1 (0%)

**Usage**

opaque.spct

**Format**

A `filter_spct` object with 4 rows and 2 variables

**Details**

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

**See Also**

Other Spectral data examples:  
`A.illuminant.spct`, `D65.illuminant.spct`, `Ler_leaf.spct`, `Ler_leaf_rflt.spct`,  
`Ler_leaf_trns.spct`, `Ler_leaf_trns_i.spct`, `black_body.spct`, `ccd.spct`, `clear.spct`, `clear_body.spct`,  
`filter_cps.mspct`, `green_leaf.spct`, `photodiode.spct`, `polyester.spct`, `sun.daily.data`,  
`sun.daily.spct`, `sun.data`, `sun.spct`, `white_body.spct`, `white_led.cps_spct`, `white_led.raw_spct`,  
`white_led.source_spct`, `yellow_gel.spct`

**Examples**

opaque.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.
Oper_spectra

Usage

```r
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 values, 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()`. 
peaks

**Peaks or local maxima**

Function that returns a subset of an R object with observations corresponding to local maxima.

**Usage**

```r
peaks(x, span, ignore_threshold, strict, na.rm, ...)
```

## Default S3 method:
```r
default <- function(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)
```

## S3 method for class 'numeric'
```r
numeric <- function(x, span = 5, ignore_threshold = NA, strict = TRUE, na.rm = FALSE, ...)
```

## S3 method for class 'data.frame'
```r
data.frame <- function(x, span = 5, ignore_threshold = 0, strict = TRUE, na.rm = FALSE, x.var.name = NULL, y.var.name = NULL, var.name = y.var.name, ...)
```

Examples

```r
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

refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'generic_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
var.name = NULL,
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'source_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
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,
ignore_threshold = 0,
strict = TRUE,
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,
ignore_threshold = 0,
strict = TRUE,
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,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'cps_spct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_spct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'source_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'response_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'filter_mspct'
peaks(
  x,
  span = 5,
ignore_threshold = 0,
strict = TRUE,
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,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'cps_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  ...,
refine.wl = FALSE,
method = "spline",
..., 
.parallel = FALSE,
.paropts = NULL
)

Arguments

x
an R object

span
integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use NULL for the global peak.

ignore_threshold
numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.

strict
logical 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.

Value

A subset of x with rows corresponding to local maxima.

Methods (by class)

- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
• source_spct: Method for "source_spct" objects.
• response_spct: Method for "response_spct" objects.
• filter_spct: Method for "filter_spct" objects.
• reflector_spct: Method for "reflector_spct" objects.
• cps_spct: Method for "cps_spct" objects.
• raw_spct: Method for "raw_spct" objects.
• generic_mspct: Method for "generic_mspct" objects.
• source_mspct: Method for "source_mspct" objects.
• response_mspct: Method for "cps_mspct" objects.
• filter_mspct: Method for "filter_mspct" objects.
• reflector_mspct: Method for "reflector_mspct" objects.
• cps_mspct: Method for "cps_mspct" objects.
• raw_mspct: Method for "raw_mspct" objects.

Note
Thresholds for ignoring peaks are applied after peaks are searched for, and negative threshold values can in some cases result in no peaks being returned.

See Also
Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples
peaks(sun.spct, span = 51)
peaks(sun.spct, span = NULL)
peaks(sun.spct, span = 51, refine.wl = TRUE)

peaks(sun.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


See Also


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 (energy) irradiances (W m\(^{-2}\) nm\(^{-1}\)).
- **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

```r
# 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))
```

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, by default as energy (W m-2 nm-1).
- **w.band**: waveband.
- **unit.in**: character Values recognized "photon" or "energy".
- **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 m-2 nm-1] -> [mol s-1 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()`, `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)))
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 (energy or photon) irradiances (W m\(^{-2}\) nm\(^{-1}\)) or (mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)).
- **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.
plus-.generic_spct

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

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

Arithmetic Operators

Description

Division operator for generic spectra.

Usage

```r
## 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()`
polyester.spct  Transmittance spectrum of clear polyester film

Description
A dataset containing the wavelengths at a 1 nm interval and fractional total transmittance for polyester film.

Usage
polyester.spct

Format
A filter_spct object with 611 rows and 2 variables

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

See Also

Examples
polyester.spct

print
Print a spectral object

Description
Print method for objects of spectral classes.

Usage

## S3 method for class 'generic_spct'
print(x, ..., n = NULL, width = NULL)

## S3 method for class 'generic_mspct'
print(x, ..., n = NULL, width = NULL, n.members = 10)
print.solar_time

Arguments

x  An object of one of the summary classes for spectra

... not used in current version

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

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.

Value

Returns x invisibly.

Methods (by class)

• generic_mspct:

Note

At the moment just a modified copy of dplyr::print.tbl_df.

Examples

print(sun.spct)
print(sun.spct, n = 5)

print.solar_time  Print solar time and solar date objects

Description

Print solar time and solar date objects

Usage

## S3 method for class 'solar_time'
print(x, ...)

## S3 method for class 'solar_date'
print(x, ...)
print.summary_generic_spct

Arguments

x an R object
... passed to format method

Note

Default is to print the underlying POSIXct as a solar time.

See Also

Other astronomy related functions: day_night(), format.solar_time(), format.tod_time(), is.solar_time(), print.tod_time(), solar_time(), sun_angles()

Description

A function to nicely print objects of classes "summary...spct".

Usage

## S3 method for class 'summary_generic_spct'
print(x, ...)

Arguments

x An object of one of the summary classes for spectra
... not used in current version

Examples

print(summary(sun.spct))
print.tod_time  

Print time-of-day objects

Description

Print time-of-day objects

Usage

## S3 method for class 'tod_time'
print(x, ...)

Arguments

x  
an R object

...  
passed to format method

Note

Default is to print the underlying numeric vector as a solar time.

See Also

Other astronomy related functions: day_night(), format.solar_time(), format.tod_time(), is.solar_time(), print.solar_time(), solar_time(), sun_angles()

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**

```r
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 values, sorted in ascending order.
- `s.irrad` A numeric vector with the sum of the two spectral values at each wavelength.
q2e

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)

q2e

Convert photon-based quantities into energy-based quantities

Description

Function that converts spectral photon irradiance (molar) into spectral energy irradiance.

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
- `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.

Methods (by class)

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

See Also

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

description

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

Usage

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

## Default S3 method:

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

## S3 method for class 'source_spct'

```r
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 = ifelse(naming != "none", "[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 = ifelse(naming != "none", "[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
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

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 [mol J⁻¹].

Methods (by class)

- default: Default for generic function
- source_spct: Method for source_spct objects
- source_mspct: Calculates photon:energy ratio 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.m ult=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_ratio()

Examples

qe_ratio(sun.spct, new_waveband(400,700))
Photon fluence (i.e. quantum irradiance) for one or more waveband of a light source spectrum.

Usage

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

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

## S3 method for class 'source_spct'
```r
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'

\texttt{q\_fluence(}
  \texttt{spct,}
  \texttt{w.band = NULL,}
  \texttt{exposure.time,}
  \texttt{scale.factor = 1,}
  \texttt{wb.trim = getOption("photobiology.waveband.trim", default = TRUE),}
  \texttt{use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),}
  \texttt{use.hinges = NULL,}
  \texttt{allow.scaled = FALSE,}
  \texttt{naming = "default",}
  \texttt{...,
  attr2tb = NULL,}
  \texttt{idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL

Arguments

\begin{itemize}
  \item \texttt{spct} an R object.
  \item \texttt{w.band} a list of waveband objects or a waveband object
  \item \texttt{exposure.time} lubridate::duration object.
  \item \texttt{scale.factor} numeric vector of length 1, or length equal to that of \texttt{w.band}. Numeric multiplier applied to returned values.
  \item \texttt{wb.trim} logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
  \item \texttt{use.cached.mult} logical indicating whether multiplier values should be cached between calls.
  \item \texttt{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.
  \item \texttt{allow.scaled} logical indicating whether scaled or normalized spectra as argument to \texttt{spct} are flagged as an error.
  \item \texttt{...} other arguments (possibly ignored).
  \item \texttt{naming} character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
  \item \texttt{attr2tb} character vector, see \texttt{add.attr2tb} for the syntax for \texttt{attr2tb} passed as is to formal parameter \texttt{col.names}.
  \item \texttt{idx} character Name of the column with the names of the members of the collection of spectra.
  \item \texttt{.parallel} if TRUE, apply function in parallel, using parallel backend provided by foreach
  \item \texttt{.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.
\end{itemize}
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.

Methods (by class)

• default: Default for generic function
• source_spct: Calculate photon fluence from a source_spct object and the duration of the
  exposure
• source_mspct: Calculates photon (quantum) fluence 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 wave-
lenghts 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_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,
## 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",
  ...
)

## 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.
w.b.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

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. 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]

Methods (by class)

- default: Default for generic function
- source_spect: Calculates photon irradiance from a source_spect object.
- source_mspct: Calculates photon (quantum) irradiance 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 irradiance functions: e_fluence(), e_irrad(), fluence(), irrad(), q_fluence()
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,
  naming = "short",
  name.tag = ifelse(naming != "none", "[q:q]", ""),
  ...
)

## 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,
  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) 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)

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.

Value

In the case of methods for individual spectra, a numeric vector of adimensional values giving a photon ratio between integrated photon irradiances for pairs of wavebands, with name attribute set to the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used, with "(q:q)" appended. A data.frame in the case of collections of spectra, containing one column for each ratio 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.

Ratio definitions are "assembled" from the arguments passed to `w.band.num` and `w.band.denom`
If both arguments are of equal length, then the wavebands are paired to obtain as many ratios 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)
- default: Default for generic function
- source_spct: Method for `source_spct` objects
- 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 wave-
lengths 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()`, `qe_ratio()`

Examples

```r
q_ratio(sun.spct, new_waveband(400,500), new_waveband(400,700))
```

---

**q_response**  
Photon-based photo-response

**Description**
This function returns the mean response for a given waveband and a response spectrum.

**Usage**
```r
def q_response(specta, 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",
  ...

Arguments

spct an R object.
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)

• default: Default method for generic function

• response_spct: Method for response spectra.

• 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

```r
q_response(ccd.spct, new_waveband(200,300))
q_response(photodiode.spct)
```

---

**rbindspct**

*Row-bind spectra*

Description

A wrapper on `dplyr::rbind.fill` that preserves class and other attributes of spectral objects.

Usage

```r
rbindspct(
  l,
  use.names = TRUE,
  fill = TRUE,
  idfactor = TRUE,
  attrs.source = NULL
)
```

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 merged.

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 '1' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct),
                   idfactor = "ID")
head(spct)
class(spct)

reflectance

## 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,
.parallelopts = 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,
.parallelopts = 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"

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 wave-
bands.

... 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.
relative_AM

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)

- **default**: Default for generic function
- **reflector_spct**: Specialization for reflector_spct
- **object_spct**: Specialization for object_spct
- **reflector_mspct**: Calculates reflectance from a reflector_mspct
- **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)))`

---

**relative_AM**

**Relative Air Mass (AM)**

**Description**

Approximate relative air mass (AM) from sun elevation or sun zenith angle.

**Usage**

```R
relative_AM(elevation.angle = NULL, zenith.angle = NULL, occluded.value = NA)
```
Arguments

elevation.angle, zenith.angle
numeric vector Angle in degrees for the sun position. An argument should be passed to one and only one of elevation_angle or zenith_angle.

occluded.value numeric Value to return when elevation angle is negative (sun below the horizon).

Details

This is an implementation of equation (3) in Kasten and Young (1989). This equation is only an approximation to the tabulated values in the same paper. Returned values are rounded to three significant digits.

Note

Although relative air mass is not defined when the sun is not visible, returning a value different from the default NA might be useful in some cases.

References


Examples

relative_AM(c(90, 60, 30, 1, -10))
relative_AM(c(90, 60, 30, 1, -10), occluded.value = Inf)
relative_AM(zenith.angle = 0)

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


See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), peaks(), spikes(), valleys(), wls_at_target()

Examples

```r
# 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]
```
**Description**

Calculate average photon- or energy-based photo-response.

**Usage**

```r
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(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 "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. 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.
- **...**: 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)

- default: Default for generic function
- 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()
Usage

\begin{verbatim}
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)
\end{verbatim}

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. \texttt{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

\begin{verbatim}
Rfr_from_n(0:90)
Rfr_from_n(0:90, p_fraction = 1)
Rfr_from_n(0:90, n = 1.333) # water
\end{verbatim}

\begin{verbatim}
rgb_spct
\end{verbatim}

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

```r
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".

### 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.

This function alters `x` itself by reference. If `x` is not a generic_spct object, `x` is not modified.

### See Also

Other set and unset spectral class functions: `setGenericSpct()`

### Examples

```r
my.spct <- sun.spct
removed <- rmDerivedSpct(my.spct)
removed
class(sun.spct)
class(my.spct)
```
### 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

```r
## 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

```r
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`
setBSWFUsed

Set the "bswf.used" attribute

Description

Function to set by reference the "time.unit" attribute of an existing source_spct object.

Usage

setBSWFUsed(x, bswf.used = c("none", "unknown"))

Arguments

x
a source_spct object

bswf.used
a character string, either "none" or the name of a BSWF

Value

x

Note

This function 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. time.unit = "hour" is currently not fully supported.

See Also

Other BSWF attribute functions: getBSWFUsed()

setFilterProperties

Set the "filter.properties" attribute

Description

Function to set by reference the "filter.properties" attribute of an existing filter_spct object.
setFilterProperties

Usage

setFilterProperties(
  x,
  filter.properties = NULL,
  pass.null = FALSE,
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA
)

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 to filter.properties is NULL.
  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".

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.

See Also

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

```r
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)
```

Description

Convert an R object into a spectrum object.

Usage

```r
setGenericSpct(x, multiple.wl = 1L, idfactor = NULL)
```

```r
setCalibrationSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)
```

```r
setRawSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)
```

```r
setCpsSpct(
  x,
  time.unit = "second",
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)
```

```r
setFilterSpct(
```
setGenericSpect(x, Tfr.type = c("total", "internal"),
Rfr.constant = NA_real_,
thickness = NA_real_,
attenuation.mode = NA,
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL)

setReflectorSpect(x, Rfr.type = c("total", "specular"),
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL)

setObjectSpect(x, Tfr.type = c("total", "internal"),
Rfr.type = c("total", "specular"),
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL)

setResponseSpect(x, time.unit = "second", multiple.wl = 1L,
idfactor = NULL)

setSourceSpect(x, time.unit = "second",
bswf.used = c("none", "unknown"),
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL)

setChromaSpect(x, multiple.wl = 1L,
idfactor = NULL)

Arguments

x data.frame, list or generic_spect 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 logitudinally (required if multiple.wl > 1).
strict.range logical Flag indicating whether off-range values result in an error instead of a warning.
### 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".
- `setReflectorSpct`: Set class of a an object to "reflector_spct".
- `setObjectSpct`: Set class of an object to "object_spct".
- `setResponseSpct`: Set class of a an object to "response_spct".
- `setSourceSpct`: Set class of an object to "source_spct".
- `setChromaSpct`: Set class of an object to "chroma_spct".

### Note

This method alters x itself by reference and in addition returns x invisibly.

For non-diffusing materials like glass an approximate `Rfr.constant` value can be used to interconvert "total" and "internal" transmittance values. Use NA if not known, or not applicable, e.g., for materials subject to internal scattering.

### See Also

Other set and unset spectral class functions: `rmDerivedSpct()`

### Examples

```r
g <- data.frame(w.length = 300:309, s.e.irrad = rep(100, 10))
is.source_spct(g)
setSourceSpct(g)
is.source_spct(g)
```
**setHowMeasured**

Set the "how.measured" attribute

**Description**

Function to set by reference the "how.measured" attribute of an existing generic_spct or derived-class object.

**Usage**

```
setHowMeasured(x, how.measured)
```

```
how_measured(x) <- value
```

**Arguments**

- **x** : a generic_spct object
- **how.measured, value** : a list

**Value**

`x`

**Note**

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.

**See Also**


**Examples**

```
my.spct <- sun.spct
how_measured(my.spct)
how_measured(my.spct) <- "simulated with a radiation transfer model"
how_measured(my.spct)
```
**setIdFactor**  
*Set the "idfactor" attribute*

**Description**
Function to set by reference the "idfactor" attribute of an existing generic_spct or an object of a class derived from generic_spct.

**Usage**
```r
setIdFactor(x, idfactor)
```

**Arguments**
- `x` a generic_spct object
- `idfactor` character The name of a factor identifying multiple spectra stored longitudinally.

**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()`

---

**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.

**Usage**
```r
setInstrDesc(x, instr.desc)
```

**Arguments**
- `x` a generic_spct object
- `instr.desc` a list
**setInstrSettings**

Value

x

Note

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.

See Also


**Description**

Function to set by reference the "what.measured" attribute of an existing generic_spct or derived-class object.

**Usage**

```
setInstrSettings(x, instr.settings)
```

**Arguments**

- x  
  a generic_spct object  
- instr.settings  
  a list

**Value**

x

**Note**

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.

**See Also**

setMultipleWl  

Set the "multiple.wl" attribute

Description

Function to set by reference the "multiple.wl" attribute of an existing generic_spct or an object of a class derived from generic_spct.

Usage

setMultipleWl(x, multiple.wl = NULL)

Arguments

x  
a generic_spct object

multiple.wl  
numeric >= 1 If multiple.wl is NULL, the default, the attribute is not modified if it is already present and valid, and set to 1 otherwise.

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. If multiple.wl

See Also

Other multiple.wl attribute functions: getMultipleWl()

setNormalized  

Set the "normalized" attribute

Description

Function to write the "normalized" attribute of an existing generic_spct object.

Usage

setNormalized(x, norm = FALSE)

setNormalised(x, norm = FALSE)
Arguments

- `x`: a generic_spect object
- `norm`: numeric or logical

Note

If `x` is not a generic_spect object, `x` is not modified.

`setNormalised()` is a synonym for this `setNormalized()` method.

See Also

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setScaled()`

---

**setRfrType** 

Set the "Rfr.type" attribute

Description

Function to set by reference the "Rfr.type" attribute of an existing reflector_spect or object_spect object.

Usage

`setRfrType(x, Rfr.type = c("total", "specular"))`

Arguments

- `x`: a reflector_spect or an object_spect object
- `Rfr.type`: a character string, either "total" or "specular"

Value

- `x`

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a reflector_spect or object_spect 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

Other Rfr attribute functions: `getRfrType()`
setScaled

Examples

```r
my.spct <- reflector_spct(w.length = 400:409, Rfr = 0.1)
getRfrType(my.spct)
setRfrType(my.spct, "specular")
getRfrType(my.spct)
```

Description

Function to write the "scaled" attribute of an existing generic_spct object.

Usage

```r
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)

• default: Default for generic function
• generic_spct: Specialization for generic_spct
• summary_generic_spct: Specialization for summary_generic_spct
• 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()`

---

**setTfrType**

Set the "Tfr.type" attribute

Description

Function to set by reference the "Tfr.type" attribute of an existing filter_spct or object_spct object

Usage

```r
setTfrType(x, Tfr.type = c("total", "internal"))
```

Arguments

- `x`: a filter_spct or an object_spct object
- `Tfr.type`: a character string, either "total" or "internal"

Value

`x`

Note

This function 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

Other Tfr attribute functions: `getTfrType()`
**setTimeUnit**

**Examples**

```r
my.spct <- polyester.spct
getTfrType(my.spct)
setTfrType(my.spct, "internal")
getTfrType(my.spct)
```

**Description**

Function to set by reference the "time.unit" attribute

**Usage**

```r
setTimeUnit(
x, 
time.unit = c("second", "hour", "day", "exposure", "none"), 
override.ok = FALSE
)
```

**Arguments**

- `x`: a source_spct object
- `time.unit`: a character string, either "second", "hour", "day", "exposure" or "none", or a lubridate::duration
- `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()`, `convertTfrType()`, `convertThickness()`, `convertTimeUnit()`, `getTimeUnit()`
Examples

```r
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**

Function to set by reference the "what.measured" attribute of an existing **generic_spct** or derived-class object.

**Usage**

```r
setWhatMeasured(x, what.measured)
what_measured(x) <- value
```

**Arguments**

- `x`: a **generic_spct** object
- `what.measured`, `value`: a list

**Value**

`x`

**Note**

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.

**See Also**


**Examples**

```r
my.spct <- sun.spct
what_measured(my.spct)
what_measured(my.spct) <- "Sun"
what_measured(my.spct)
```
setWhenMeasured

Set the "when.measured" attribute

Description

Function to set by reference the "when" attribute of an existing generic_spct or an object of a class derived from generic_spct.

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 'generic_mspct'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)

Arguments

x a generic_spct object
when.measured, value POSIXct to add as attribute, or a list of POSIXct.
... Allows use of additional arguments in methods for other classes.

Value

x

Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct
setWhereMeasured

Note
This method 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. If when 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).

See Also

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)

setWhereMeasured

Set the "where.measured" attribute

Description
Function to set by reference the "where.measured" attribute of an existing generic_spct or an object of a class derived from generic_spct.

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 'generic_mspct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)
Arguments
- x: a generic_spct object
- where.measured: A one row data.frame such 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

Methods (by class)
- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

Note
This method 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. If where 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 expected passing NULL as argument for where.measured unsets the attribute. Method for collections of spectra recycles the location information only if it is of length one.

See Also

Examples
```r
my.spct <- sun.spct
where_measured(my.spct) <- data.frame(lon = 0, lat = -60)
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 members as a target set of class names.

**Usage**

```r
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**

```r
## 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`
### Arithmetic Operators

**Description**

Division operator for generic spectra.

**Usage**

```r
## 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**

```r
smooth_spct(x, method, strength, ...)
```

**Default S3 method:**

```r
smooth_spct(x, method, strength, ...)
```

**S3 method for class 'source_spct'**

```r
smooth_spct(x, method = "custom", strength = 1, na.rm = FALSE, ...)
```

**S3 method for class 'filter_spct'**

```r
smooth_spct(x, method = "custom", strength = 1, na.rm = FALSE, ...)
```
smooth_spct

## S3 method for class 'reflector_spct'
smooth_spct(x, method = "custom", strength = 1, na.rm = FALSE, ...)

## S3 method for class 'response_spct'
smooth_spct(x, method = "custom", strength = 1, na.rm = FALSE, ...)

## S3 method for class 'generic_mspct'
smooth_spct(x, method = "custom", strength = 1, na.rm = FALSE, ...)

### Arguments

- **x**: an R object.
- **method**: a character string "custom", "lowess", "supsmu".
- **strength**: numeric value to adjust the degree of smoothing.
- **...**: other parameters passed to the underlying smoothing functions.
- **na.rm**: logical A value 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)

- **default**: Default for generic function
- **source_spct**: Smooth a source spectrum
- **filter_spct**: Smooth a filter spectrum
- **reflector_spct**: Smooth a reflector spectrum
- **response_spct**: Smooth a response spectrum
- **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.

### Examples

```r
my.spct <- clip_wl(sun.spct, c(400, 500))
smooth_spct(my.spct)
smooth_spct(my.spct, method = "supsmu", strength = 4)
```
**solar_time**

---

**solar_time**

*Local solar time*

### Description

`solar_time` computes from a time and geocode, the time of day expressed in seconds since midnight. `solar_date` returns the same instant in time as a date-time object. Solar time is useful when we want to plot data according to the local solar time of day, irrespective of the date. Solar date is useful when we want to plot a time series stretching for several days using the local solar time but distinguishing between days.

### Usage

```r
solar_time(
  time = lubridate::now(),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  unit.out = "time"
)
```

### Arguments

- **time**: POSIXct Time, any valid time zone (TZ) is allowed, default is current time
- **geocode**: data frame with variables lon and lat as numeric values (degrees).
- **unit.out**: character string. One of "datetime", "hour", "minute", or "second".

### Value

For `solar_time()` numeric value in seconds from midnight but with an additional class attribute "solar.time".

### Warning!

Returned values are computed based on the time zone of the argument for parameter time. In the case of solar time, this timezone does not affect the result. However, in the case of solar dates the date part may be off by one day, if the time zone does not match the coordinates of the geocode value provided as argument.

### Note

The algorithm is approximate, it calculates the difference between local solar noon and noon in the time zone of `time` and uses this value for the whole day when converting times into solar time. Days are not exactly 24 h long. Between successive days the shift is only a few seconds, and this leads to a small jump at midnight.

### See Also

Other astronomy related functions: `day_night()`, `format.solar_time()`, `format.tod_time()`, `is.solar_time()`, `print.solar_time()`, `print.tod_time()`, `sun_angles()`
Examples

```r
BA.geocode <- data.frame(lon = -58.38156, lat = -34.60368, address = "Buenos Aires, Argentina")
sol_t <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"), BA.geocode)

sol_t
class(sol_t)

sol_d <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"), BA.geocode, unit.out = "datetime")

sol_d
class(sol_d)
```

source_spct

**Spectral-object constructor**

Description

These functions can be used to create spectral objects derived from `generic_spct`. They take as arguments numeric vectors for the data character scalars for attributes, and a logical flag.

Usage

```r
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,
  ...
)
```

```r
calibration_spct(
  w.length = NULL,
  irrad.mult = NA_real_,
  comment = NULL,
  instr.desc = NA,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)
```
source_spct

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"),
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

filter_spct(
  w.length = NULL,
  Tfr = NULL,
  Tpc = NULL,
  Afr = NULL,
  A = NULL,
source_spct

Tfr.type = c("total", "internal"),
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,
  ...
)

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
source_spct

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].
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.
idfactor character Name of factor distinguishing multiple spectra when stored logitudinally (required if multiple.wl > 1).
... other arguments passed to tibble()
irrad.mult numeric vector with multipliers for each detector pixel.
instr.desc a list
counts numeric vector with raw counts expressed per scan
instr.settings a list
cps numeric vector with linearized raw counts expressed per second
s.e.response numeric vector with spectral energy irradiance in W m⁻² nm⁻¹ or J d⁻¹ m⁻² nm⁻¹
s.q.response numeric vector with spectral photon irradiance in mol s⁻¹ m⁻² nm⁻¹ or mol d⁻¹ m⁻² nm⁻¹
Tfr numeric vector with spectral transmittance as fraction of one
Tpc numeric vector with spectral transmittance as percent values
Afr numeric vector of absorptance as fraction of one
A numeric vector of absorbance values (log10 based a.u.)
Tfr.type character string indicating whether transmittance and absorptance values are “total” or “internal” values
Rfr numeric vector with spectral reflectance as fraction of one
Rpc numeric vector with spectral reflectance as percent values
Rfr.type character A string, either "total" or "specular".
x, y, z numeric colour coordinates

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.
Note

The functions can be used to add only one spectral quantity to a spectral object. Some of the functions have different arguments, for the same quantity expressed in different units. An actual parameter can be supplied to only one of these formal parameters in a given call to any of these functions.

"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.

See Also

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()`

---

**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**

```r
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`. 
spct_classes

Function that returns a vector containing the names of spectra classes.

Description

Function that returns a vector containing the names of spectra classes.

Usage

spct_classes()

Value

A character vector of class names.

Examples

spct_classes()

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"),
    ...
spikes

## 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 '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 'cps_mspct' 
spikes(
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.

Value

A subset of `x` with rows corresponding to spikes.
Methods (by class)

- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct" objects.
- cps_spct: Method for "cps_spct" objects.
- raw_spct: Method for "raw_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "cps_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct" objects.
- cps_mspct: Method for "cps_mspct" objects.
- 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_pixels(), valleys(), wls_at_target()

Examples

spikes(sun.spct)
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,
Arguments

\[ x \quad \text{data frame} \]

\[ \text{member.class} \quad \text{character Class of the collection members} \]

\[ \text{spct.data.var} \quad \text{character Name of the spectral data argument in the object constructor for member.class} \]

\[ \text{w.length.var} \quad \text{character Name of column containing wavelength data in nanometres} \]

\[ \text{idx.var} \quad \text{character Name of column containing data to be copied unchanged to each spct object} \]

\[ \text{ncol} \quad \text{integer Number of 'virtual' columns in data} \]

\[ \text{byrow} \quad \text{logical If ncol > 1 how to read in the data} \]

\[ \ldots \quad \text{additional named arguments passed to the member constructor function.} \]
split_bands  

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(), 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.spt, 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

```r
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.
split_irradiance

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

```r
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 (energy or photon) irradiances (W m-2 nm-1) or (mol s-1 m-2 nm-1).
- `cut.w.length`: numeric Vector of wavelengths (nm).
- `unit.out`: character Allowed values "energy", and "photon", or its alias "quantum".
- `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 irradiances with no change in scale factor: [W m-2 nm-1] -> [mol s-1 m-2] or [mol s-1 m-2 nm-1] -> [mol s-1 m-2] or relative values (as fraction of one if scale == "relative" or percentages if 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.
split_photon_irradiance

Examples

```r
with(sun.data,
    split_irradiance(w.length, s.e.irrad,
    cut.w.length = c(300, 400, 500, 600, 700),
    unit.out = "photon"))
```

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

```r
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.
spread

Value

A numeric vector of photon irradiances with no change in scale factor: \([W \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow [\text{mol s}^{-1} \text{ m}^{-2}]\), \([\text{mol s}^{-1} \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow [\text{mol s}^{-1} \text{ m}^{-2}]\) 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: \(\text{as_energy}(), \text{as_quantum_mol}(), \text{calc_multipliers}(), \text{div_spectra}(), \text{energy_irradiance}(), \text{energy_ratio}(), \text{insert_hinges}(), \text{integrate_xy}(), \text{interpolate_spectrum}(), \text{irradiance}(), \text{l_insert_hinges}(), \text{oper_spectra}(), \text{photon_irradiance}(), \text{photon_ratio}(), \text{photons_energy_ratio}(), \text{prod_spectra}(), \text{s_e_irrad2rgb}(), \text{split_energy_irradiance}(), \text{subt_spectra}(), \text{sum_spectra}(), \text{trim_tails}(), \text{v_insert_hinges}(), \text{v_replace_hinges}()\)

Examples

\[
\begin{align*}
\text{with(sun.data,} \\
\text{split_photon_irradiance(w.length, s.e.irrad,} \\
\text{cut.w.length = c(300, 400, 500, 600, 700)))} \\
\text{with(sun.data,} \\
\text{split_photon_irradiance(w.length, s.e.irrad))}
\end{align*}
\]

spread \hspace{1cm} Expanse

Description

A function that returns the expanse (max(x) - min(x)) for R objects.

Usage

\[
\begin{align*}
\text{spread(x, ...)} \\
\text{wl_expanse(x, ...)} \\
\text{expanse(x, ...)}
\end{align*}
\]

## 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 in nm. For any other R object, according to available definitions of min and max.

Methods (by class)

- **default**: Default method for generic function
- **numeric**: Method for "numeric"
- **waveband**: Method for "waveband"
- **generic_spct**: Method for "generic_spct"
- **generic_mspct**: Method for "generic_mspct" objects.

Examples

expanse(10:20)
expanse(sun.spct)
w1_expanse(sun.spct)
expanse(sun.spct)
Description

Return subsets of spectra stored in class generic_spct or derived from it.

Usage

```r
## 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 index 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

```r
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 spectral objects of the corresponding class. For data frames converting numeric columns other than wavelength into individual spectral objects.

Usage

subset2mspct(
  x, 
  member.class = NULL, 
  idx.var = attr(x, "idfactor"), 
  drop.idx = TRUE, 
  ncol = 1, 
  byrow = FALSE, 
  ...
)

Arguments

x a generic_spct object or a derived class, or a data frame
member.class character string
idx.var character Name of column containing data to be copied unchanged to each spct object
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 comments which are merged.

Note

A non-null value for member.class is mandatory only when x is a data frame.
subt_spectra

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()

---

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

```r
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.
summary

Value

- **w.length**: A numeric vector with the wavelengths (nm) obtained by "fusing" `w.length1` and `w.length2`. `w.length` contains all the unique values, 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

```r
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

### Summary of a spectral object

**Description**

Methods of generic function `summary` for objects of spectral classes.

**Usage**

```r
# S3 method for class 'generic_spct'
summary(object, maxsum = 7, digits = max(3, getOption("digits") - 3), ...)
```

**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

**Value**

A summary object matching the class of `object`. 
sum_spectra

Examples

```r
summary(sun.spct)
```

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

```r
summary_spct_classes()
```

Value

A character vector of class names.

sum_spectra

Add two spectra

Description

Merge wavelength vectors of 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 a 'parallel' operation between two spectra.

Usage

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

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 values, 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.daily.data  

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.data

Format
A data.frame object with 511 rows and 3 variables

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)

Author(s)
Anders K. Lindfors (data)

References

See Also

Examples
sun.daily.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

Format
A source_spct object with 511 rows and 3 variables

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)

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

See Also
Examples

sun.daily.spct

sun.data  *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.data

Format

A *data.frame* object with 508 rows and 3 variables

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)

Author(s)

Anders K. Lindfors (data)

References


See Also


Examples

sun.data
**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

**Format**

A `source_spct` object with 508 rows and 3 variables.

**Details**

- w.length (nm), range 293 to 800 nm.
- s.e.irrad (W m$^{-2}$ nm$^{-1}$)
- s.q.irrad (mol m$^{-2}$ nm$^{-1}$)

**Author(s)**

Anders K. Lindfors (data)

**References**


**See Also**


**Examples**

sun.spct
**sun_angles**

**Solar angles**

**Description**

This function returns the solar angles at a given time and location.

**Usage**

```r
sun_angles(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_angles_fast(time, tz, geocode, use.refraction)

sun_elevation(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_zenith_angle(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_azimuth(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)
```

**Arguments**

- **time**: A "vector" of POSIXct Time, with any valid time zone (TZ) is allowed, default is current time.
- **tz**: character string indicating time zone to be used in output.
- **geocode**: data frame with variables lon and lat as numeric values (degrees), nrow > 1, allowed.
- **use.refraction**: logical Flag indicating whether to correct for fraction in the atmosphere.
Details

This function is an implementation of Meeus equations as used in NOAAs on-line web calculator, which are precise and valid for a very broad range of dates (years -1000 to 3000 at least). The apparent solar elevations near sunrise and sunset are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions for the atmosphere. The computation is not defined for latitudes 90 and -90 degrees, i.e. exactly at the poles.

In the current implementation functions `sun_azimuth`, `sun_elevation`, and `sun_zenith_angle` are wrappers on `sun_angles`, so if more than one angle is needed it is preferable to directly call `sun_angles` as it will be faster.

Value

A data.frame with variables time (in same TZ as input), TZ, solartime, longitude, latitude, address, azimuth, and elevation. If a data frame with multiple rows is passed to `geocode` and a vector of times longer than one is passed to `time`, sun position for all combinations of locations and times are returned are returned by `sun_angles`. In contrast, convenience functions returning a vector.

Note

There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function `astrocalc4r` in package ‘fishmethods’. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the present implementation splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates.

References


A different implementation is available at https://www.nefsc.noaa.gov/AstroCalc4R/ and in R package `astrocalc4r`. In 'fishmethods' (= 1.11-0) there is a bug in function `astrocalc4r()` that affects sunrise and sunset times.

An interactive web page using the same algorithms is available at https://www.esrl.noaa.gov/gmd/grad/solcalc/. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

Other astronomy related functions: `day_night()`, `format.solar_time()`, `format.tod_time()`, `is.solar_time()`, `print.solar_time()`, `print.tod_time()`, `solar_time()`

Examples

library(lubridate)
sun_angles()
sun_azimuth()
sun_elevation()
sun_zenith_angle()
sun_angles(ymd_hms("2014-09-23 12:00:00"))

sun_angles(ymd_hms("2014-09-23 12:00:00"),
    geocode = data.frame(lat=60, lon=0))
sun_angles(ymd_hms("2014-09-23 12:00:00") + minutes((0:6) * 10))

---

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.
s_mean

Note

Very heavily modified from Chad Eliason’s <cme16@zips.uakron.edu> spec2rgb function in package Pavo.

References


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

A method to compute the mean of values across members of a collections of spectra. Computes the mean at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_mean(x, trim, na.rm, ...)

## Default S3 method:
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, ...)

**Arguments**

**x**  
An R object. Currently this package defines methods for collections of spectral objects.

**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.

**Value**

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spect", containing the mean spectrum.

**Methods (by class)**

- default:
- source_mspct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- calibration_mspct:

**Note**

Trimming of extreme values 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.

**See Also**

See mean for the mean() method used for the computations.
Mean and standard error from collection of spectra

Description

A method to compute the mean of values across members of a collections of spectra. Computes the mean at each wavelength across all the spectra in the collection returning a spectral object.

Usage

\[
\text{s\_mean\_se}(x, \text{na.rm}, \text{mult}, \ldots)
\]

### Default S3 method:
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

### S3 method for class 'filter\_mspec'
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

### S3 method for class 'source\_mspec'
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

### S3 method for class 'response\_mspec'
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

### S3 method for class 'reflector\_mspec'
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

### S3 method for class 'calibration\_mspec'
\[
\text{s\_mean\_se}(x, \text{na.rm} = \text{FALSE}, \text{mult} = 1, \ldots)
\]

Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **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.

Value

If \(x\) is a collection spectral of objects, such as a "filter\_mspec" object, the returned object is of same class as the members of the collection, such as "filter\_spct", containing the mean spectrum.
Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:

Note
Trimming of extreme values 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.

See Also
See mean for the mean() method used for the computations.

---

s_median

Median of a collection of spectra

Description
A method to compute the median of values across members of a collections of spectra. Computes the median at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_median(x, na.rm, ...)

## Default S3 method:
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, ...)
Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

Value

If `x` is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the median spectrum.

Methods (by class)

- default:
- source_mspct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- calibration_mspct:

Note

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.

See Also

See `median` for the `median()` method used for the computations.

### Description

A method to compute the product of values across members of a collections of spectra. Computes the product at each wavelength across all the spectra in the collection returning a spectral object.
Usage

s_prod(x, na.rm, ...)

## Default S3 method:
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, ...)

Arguments

x An R object. Currently this package defines methods for collections of spectral objects.

na.rm logical. A value indicating whether NA values should be stripped before the computation proceeds.

... Further arguments passed to or from other methods.

Value

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the product of the spectra.

Methods (by class)

• default:
• source_mspct:
• response_mspct:
• filter_mspct:
• reflector_mspct:
• calibration_mspct:

Note

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.
A product of spectral irradiance or spectral response is no longer a well defined physical quantity, and these product operations return an object of class generic_spct.

See Also

See prod for the prod() method used for the computations.

s_range

Range of a collection of spectra

Description

A method to compute the range of values across members of a collections of spectra. Computes the max and min at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_range(x, na.rm, ...)

## Default S3 method:
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, ...)

Arguments

x An R object. Currently this package defines methods for collections of spectral objects.

na.rm logical. A value indicating whether NA values should be stripped before the computation proceeds.

... Further arguments passed to or from other methods.
Value

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the mean spectrum.

Methods (by class)

• default:
• filter_mspct:
• source_mspct:
• response_mspct:
• reflector_mspct:
• calibration_mspct:

Note

Trimming of extreme values 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.

See Also

See Extremes details on the min() and max() methods used for the computations.

s_sd

Standard Deviation of a collection of spectra

Description

A method to compute the standard deviation of values across members of a collections of spectra. Computes the standard deviation at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_sd(x, na.rm, ...)

## Default S3 method:
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 'response_mspct'
s_sd(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_sd(x, na.rm = FALSE, ...)

Arguments

x
An R object. Currently this package defines methods for collections of spectral objects.

na.rm
logical. A value indicating whether NA values should be stripped before the computation proceeds.

...
Further arguments passed to or from other methods.

Value

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of class "generic_spct", containing the standard deviation among the spectra at each wavelength in a column with name ending in ".sd".

Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:

Note

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.

See Also

See sd for details about sd() methods for other classes.

---

**s_se**

*Standard Error of a collection of spectra*

Description

A method to compute the standard error of values across members of a collections of spectra. Computes the standard error at each wavelength across all the spectra in the collection returning a spectral object.
Usage

s_se(x, na.rm, ...)

## Default S3 method:
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, ...)

Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

Value

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of class "generic_spct", containing the standard error among the spectra at each wavelength in a column with name ending in ".se".

Methods (by class)

- default:
- source_mspct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- calibration_mspct:
Note

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.

---

**s_sum**

*Sum from collection of spectra*

**Description**

A method to compute the sum of values across members of a collections of spectra. Computes the sum at each wavelength across all the spectra in the collection returning a spectral object.

**Usage**

```r
s_sum(x, na.rm, ...)
```

## Default S3 method:

```r
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'filter_mspct'

```r
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'source_mspct'

```r
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'response_mspct'

```r
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'reflector_mspct'

```r
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'calibration_mspct'

```r
s_sum(x, na.rm = FALSE, ...)
```

**Arguments**

- **x**
  
  An R object. Currently this package defines methods for collections of spectral objects.

- **na.rm**
  
  logical. A value indicating whether NA values should be stripped before the computation proceeds.

- **...**
  
  Further arguments passed to or from other methods.

**Value**

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spect", containing the sum of the spectra.
Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:

Note

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.

A sum of transmittances or reflectances is no longer a well defined physical quantity, and these sum operations return an object of class generic_spct.

See Also

See sum for the sum() method used for the computations.

---

s_var Variance of a collection of spectra

Description

A method to compute the variance of values across members of a collections of spectra. Computes the variance at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_var(x, na.rm, ...)

## Default S3 method:
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, ...)
's_var

## S3 method for class 'calibration_mspct'
s_var(x, na.rm = FALSE, ...)

### Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **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

Variance method for collections of spectra. Computes the variance at each wavelength across all the spectra in the collection.

### Value

If x is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of class "generic_spct", containing the variance among the spectra at each wavelength in a column with name ending in ".var".

### Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:

### Note

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.

### See Also

See cor for details about var(), which is used for the computations.
Description

Function that converts transmittance (fraction) into 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, ...)

## S3 method for class 'filter_mspct'
T2A(
  x,
  action = "add",
  byref = FALSE,
  clean = 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

.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 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)

• default: Default method for generic function
• numeric: Method for numeric vectors
• filter_spct: Method for filter spectra
• filter_mspct: Method for collections of filter spectra

See Also

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

T2Afr

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

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,
### S3 method for class 'object_mspct'

```r
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 or 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 `Afr` 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)**

- **default**: Default method for generic function
- **numeric**: Default method for generic function
- **filter_spct**: Method for filter spectra
- **object_spct**: Method for object spectra
- **filter_mspct**: Method for collections of filter spectra
- **object_mspct**: Method for collections of object spectra
See Also

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

Examples

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

---

**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 method is most useful for plotting realistic computed colors from spectral data.

**Usage**

```r
tag(x, ...)
```

```r
## Default S3 method:
tag(x, ...)
```

```r
## 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,
  ...
)
```

```r
## 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,
  ..., 
)```
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)

- default: Default method for generic
- generic_spct: Tag one of generic_spct, and derived classes including source_spct, filter_spct, reflector_spct, object_spct, and response_spct.
- 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

```r
tag(sun.spct)
tag(sun.spct, list(A = waveband(c(300,3005))))
```

---

### Description

Increase the wavelength step in stored spectral data in featureless regions to save storage space.

#### Usage

```r
thin_wl(x, ...)
```

- **Default S3 method:**
  ```r
  thin_wl(x, ...)
  ```

- **S3 method for class 'generic_spct'**
  ```r
  thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)
  ```

- **S3 method for class 'source_spct'**
  ```r
  thin_wl(
    x,
    max.wl.step = 10,
    max.slope.delta = 0.001,
    unit.out = getOption("photobiology.radiation.unit", default = "energy"),
    ...
  )
  ```

- **S3 method for class 'response_spct'**
  ```r
  thin_wl(
    x,
    max.wl.step = 10,
    max.slope.delta = 0.001,
    unit.out = getOption("photobiology.radiation.unit", default = "energy"),
    ...
  )
  ```

- **S3 method for class 'filter_spct'**
  ```r
  thin_wl(
    x,
    max.wl.step = 10,
    max.slope.delta = 0.001,
    qty.out = getOption("photobiology.filter_qty", default = "transmittance"),
    ```
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.
- `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)

- default: Default for generic function
- generic_spct:
- source_spct:
- response_spct:
- filter_spct:
- reflector_spct:
- raw_spct:
- cps_spct:
- object_spct:
- chroma_spct:
- calibration_spct:
- generic_mspct:
- chroma_mspct:
- 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.

See Also

Other experimental utility functions: uncollect()
Examples

nrow(yellow_gel.spct)
w_l_stepsize(yellow_gel.spct)

thinned.spct <- thin_wl(yellow_gel.spct)
nrow(thinned.spct)
w_l_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.
transmittance

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 = getOption("photobiology.use.hinges", default = 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),
  ...
)

## 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 "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
wb.trim       logical Flag indicating 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.
...           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)

- default: Default method
- filter_spct: Method for filter spectra
- object_spct: Method for object spectra
- filter_mspct: Calculates transmittance from a filter_mspct
- 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

```r
transmittance(polyester.spct, waveband(c(280, 315)))
transmittance(polyester.spct, waveband(c(315, 400)))
transmittance(polyester.spct, waveband(c(400, 700)))
```

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

```r
## 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)
```
trimInstrDesc

Trim the "instr.desc" attribute

Description

Function to trim the "instr.desc" attribute of an existing generic_spct object, discarding all fields except for 'spectrometer.name', 'spectrometer.sn', 'bench.grating', 'bench.slit', and calibration name.

Usage

```r
trimInstrDesc(
  x,
  fields = c("time", "spectrometer.name", "spectrometer.sn", "bench.grating", "bench.slit")
)
```

Arguments

- **x**: a 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.

Value

- **x**: the trimmed spectrum object

Note

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.

See Also

trimInstrSettings

Trim the "instr.settings" attribute

Description

Function to trim the "instr.settings" attribute of an existing generic_spct object, by discarding some fields.

Usage

trimInstrSettings(x, fields = "*")

Arguments

x a 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.

Value

x

Note

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.

See Also


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).
trim_spct

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 of same class as input with its tails trimmed or expanded.

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()
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()`, `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,
  ...
)

## 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` is `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)

- **default**: Default for generic function
- **generic_spct**: Trim an object of class "generic_spct" or derived.
- **generic_mspct**: Trim an object of class "generic_mspct" or derived.
- **waveband**: Trim an object of class "waveband".
- **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.

See Also

Other trim functions: `clip_wl()`, `trim_spct()`, `trim_waveband()`
Examples

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

### tz_time_diff

**Time difference between two time zones**

**Description**

Returns the time difference in hours between two time zones at a given instant in time.

**Usage**

```r
tz_time_diff(
    when = lubridate::now(),
    tz.target = lubridate::tz(when),
    tz.reference = "UTC"
)
```

**Arguments**

- `when` : datetime A time instant
- `tz.target`, `tz.reference` : character Two time zones using names recognized by functions from package 'lubridate'

**Value**

A numeric value.

### uncollect

**Extract all members from a collection**

**Description**

Extract all members from a collection into separate objects in the parent frame of the call.
Usage

```r
uncollect(x, ...) # Default S3 method:
uncollect(x, ...)
```

```r
## S3 method for class 'generic_mspct'
uncollect(
  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)

- default: Default for generic function
- `generic_mspct`:

See Also

Other experimental utility functions: `thin_wl()`
untag

Examples

```r
my.mscpt <- source_mspct(list(sun1 = sun.spct, sun2 = sun.spct))
uncollect(my.mscpt)
ls(pattern = "*.spct")
```

untag Remove tags

Description

Remove tags from an R object if present, otherwise return the object unchanged.

Usage

```r
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)

- default: Default for generic function
- `generic_spct`: Specialization for `generic_spct`
- `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 of 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 http://www.burns-stat.com/the-options-mechanism-in-r/, section Deep End, of "The Options mechanism in R" by Patrick Burns.
**validate_geocode**

Valid a geocode

**Description**

Test validity of a geocode or ensure that a geocode is valid.

**Usage**

validate_geocode(geocode)

is_valid_geocode(geocode)

length_geocode(geocode)

na_geocode()

**Arguments**

geocode data.frame with geocode data in columns "lat", "lon", and possibly also "address".

**Details**

validate_geocode Converts to tibble, checks data bounds, converts address to character if it is not already a character vector, or add character NAs if the address column is missing.

is_valid_geocode Checks if a geocode is valid, returning 0L if not, and the number of row otherwise.

**Value**

A valid geocode stored in a tibble.

FALSE for invalid, TRUE for valid.

FALSE for invalid, number of rows for valid.

A geo_code tibble with all fields set to suitable NAs.

**Examples**

validate_geocode(NA)

validate_geocode(data.frame(lon = -25, lat = 66))

is_valid_geocode(NA)

is_valid_geocode(1L)

is_valid_geocode(data.frame(lon = -25, lat = 66))

na_geocode()
Description

Function that returns a subset of an R object with observations corresponding to local maxima.

Usage

valleys(x, span, ignore_threshold, strict, ...)

## Default S3 method:
valleys(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)

## Default S3 method:
valleys(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)

## S3 method for class 'numeric'
valleys(x, span = 5, ignore_threshold, strict = TRUE, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  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,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
)
### S3 method for class 'source_spct'
```
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  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,
  ignore_threshold = 0,
  strict = TRUE,
  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,
  ignore_threshold = 0,
  strict = TRUE,
  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,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  ```
valleys

  method = "spline",

  ...
)

## S3 method for class 'cps_spct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_spct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_mspct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  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,
ignore_threshold = 0,
strict = TRUE,
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,
ignore_threshold = 0,
strict = TRUE,
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,
ignore_threshold = 0,
strict = TRUE,
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,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
valleys

refine.wl = FALSE,
method = "spline",
..., .parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'cps_mspct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  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,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ..., .parallel = FALSE,
  .paropts = NULL
)

Arguments

x an R object

span integer A valley is defined as an element in a sequence which is smaller than all other elements within a window of width span centered at that element. Use NULL for the global peak.

ignore_threshold numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.

strict logical 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.

Value

A subset of x with rows corresponding to local minima.

Methods (by class)

- default: Default function usable on numeric vectors.
- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct".
- cps_spct: Method for "cps_spct" objects.
- raw_spct: Method for "raw_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "cps_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct".
- cps_mspct: Method for "cps_mspct" objects.
- raw_mspct: Method for "raw_mspct" objects.
See Also

Other peaks and valleys functions: `find_peaks()`, `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `wls_at_target()

Examples

valleys(sun.spct, span = 50)

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)`
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_replace_hinges()
water_vp_sat

**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_vp_sat**

*Water vapour pressure*

**Description**

Approximate water pressure in air as a function of temperature, and its inverse the calculation of dewpoint.

**Usage**

```r
water_vp_sat(
  temperature,
  over.ice = FALSE,
  method = "tetens",
  check.range = TRUE
)
```

```r
water_dp(water.vp, over.ice = FALSE, method = "tetens", check.range = TRUE)
```

```r
water_fp(water.vp, over.ice = TRUE, method = "tetens", check.range = TRUE)
```

```r
water_vp2mvc(water.vp, temperature)
```

```r
water_mvc2vp(water.mvc, temperature)
```

```r
water_vp2RH(
  water.vp,
  temperature,
  over.ice = FALSE,
  method = "tetens",
  pc = TRUE,
  check.range = TRUE
)
```

```r
water_RH2vp(
  relative.humidity,
  temperature,
  over.ice = FALSE,
  method = "tetens",
  pc = TRUE,
  check.range = TRUE
)
```

```r
c = TRUE,
check.range = TRUE
)
```

**Arguments**

- `temperature` numeric vector of air temperatures (C).
- `over.ice` logical Is the estimate for equilibrium with liquid water or with ice.
- `method` character Currently "tetens", modified "magnus", "wexler" and "goff.gratch" equations are supported.
- `check.range` logical Flag indicating whether to check or not that arguments for temperature are within the range of validity of the method used.
- `water.vp` numeric vector of water vapour pressure in air (Pa).
- `water.mvc` numeric vector of water vapour concentration as mass per volume (g m-3).
- `pc` logical flag for result returned as percent or not.
- `relative.humidity` numeric Relative humidity as fraction of 1.

**Details**

Function `water_vp_sat()` provides implementations of several well known equations for the estimation of saturation vapor pressure in air. Functions `water_dp()` and `water_fp()` use the inverse of these equations to compute the dew point or frost point from water vapour pressure in air. The inverse functions are either analytical solutions or fitted approximations. None of these functions are solved numerically by iteration.


The equations are approximations, and in spite of their different names, Tetens’ and Magnus’ equations have the same form with the only difference in the values of the parameters. However, the modified Magnus equation is more accurate as Tetens equation suffers from some bias errors at extreme low temperatures (< -40 C). In contrast Magnus equations with recently fitted values for the parameters are usable for temperatures from -80 C to +50 C over water and -80 C to 0 C over ice. The Groff Gratch equation is more complex and is frequently used as a reference in comparison as it is considered reliable over a broad range of temperatures. Wexler’s equations are computationally simpler and fitted to relatively recent data. There is little difference at temperatures in the range -20 C to +50 C, and differences become large at extreme temperatures. Temperatures outside the range where estimations are highly reliable for each equation return NA, unless extrapolation is enabled by passing `FALSE` as argument to parameter `check.range`.

The switch between equations for ice or water cannot be based on air temperature, as it depends on the presence or not of a surface of liquid water. It must be set by passing an argument to parameter `over.ice` which defaults to `FALSE`.

Tetens equation is still very frequently used, and is for example the one recommended by FAO for computing potential evapotranspiration. For this reason it is used as default here.
A numeric vector of partial pressures in pascal (P) for `water_vp_sat` and `water_mvc2vp`, a numeric vector of dew point temperatures (C) for `water_dp` and numeric vector of mass per volume concentrations (g m^-3) for `water_vp2mvc`.

**Note**

The inverse of the Groff Gratch equation has yet to be implemented.

**References**


[Equations describing the physical properties of moist air](http://www.conservationphysics.org/atmcalc/atmoclc2.pdf)

**Examples**

```r
water_vp_sat(20) # C -> Pa
water_vp_sat(temperature = c(0, 10, 20, 30, 40)) # C -> Pa
water_vp_sat(temperature = -10) # over water!!
water_vp_sat(temperature = -10, over.ice = TRUE)
water_vp_sat(temperature = 20) / 100 # C -> mbar
water_vp_sat(temperature = 20, method = "magnus") # C -> Pa
water_vp_sat(temperature = 20, method = "tetens") # C -> Pa
water_vp_sat(temperature = 20, method = "wexler") # C -> Pa
water_vp_sat(temperature = 20, method = "goff.gratch") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "magnus") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "tetens") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "wexler") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "goff.gratch") # C -> Pa
```
waveband(water.vp = 1000) # Pa -> C
waveband(water.vp = 1000, method = "magnus") # Pa -> C
waveband(water.vp = 1000, method = "wexler") # Pa -> C
waveband(water.vp = 500, over.ice = TRUE) # Pa -> C
waveband(water.vp = 500, method = "wexler", over.ice = TRUE) # Pa -> C

waveband(water.vp = 300) # Pa -> C
waveband(water.vp = 300, over.ice = TRUE) # Pa -> C

waveband(water.vp = 1500, temperature = 20) # Pa, C -> RH %
waveband(water.vp = 1500, temperature = c(20, 30)) # Pa, C -> RH %
waveband(water.vp = c(600, 1500), temperature = 20) # Pa, C -> RH %

waveband(water.vp = 1000, temperature = 20) # Pa -> g m-3
waveband(water.vp = 30, temperature = 40) # g m-3 -> Pa
waveband(water.vp = water_mvc2vp(water.mvc = 10, temperature = 30)) # g m-3 -> C

---

**waveband**

**Waveband constructor method**

**Description**

Constructor for "waveband" objects that can be used as input when calculating irradiances.

**Usage**

```r
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
)
```

```r
new_waveband(
  w.low,
  w.high,
  weight = NULL,
  SWF.e.fun = NULL,
  SWF.q.fun = NULL,
  norm = NULL,
```
waveband

SWF.norm = NULL,
hinges = NULL,
wb.name = NULL,
wb.label = wb.name
)

Arguments

x any R object on which applying the function 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 a function giving multipliers for a spectral weighting function (energy) as a function of wavelength (nm)
SWF.q.fun a function giving multipliers for a spectral weighting function (quantum) as a function of wavelength (nm)
norm a single numeric value indicating the wavelength at which the SWF should be normalized to 1.0, in nm. "NULL" means 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 the s.irrad should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0)), if NULL then interpolation will take place at both ends of the band.
wb.name character string giving the name for the waveband defined, default is NULL
wb.label character string giving the label of the waveband to be used for plotting, default is wb.name
w.low numeric value, wavelength at the short end of the band (nm)
w.high numeric value, wavelength at the long end of the band (nm)

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 (energy or photon) irradiances (W m-2 nm-1) or (mol s-1 m-2 nm-1).
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.out.num character Allowed values "energy", and "photon", or its alias "quantum".
unit.out.denom character 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 9.19 onwards use of this default does not trigger a warning, but instead is used silently.

Examples

```r
# 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**

```r
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()`

---

**Description**

Create a `generic_spect` 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.
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(), wb2tagged_spct()
white_body.spct

---

**wb_trim_as_default**  *Set computation options*

---

**Description**
Set computation related options easily.

**Usage**
```r
wb_trim_as_default(flag = TRUE)
```
```r
use_cached_mult_as_default(flag = TRUE)
```

**Arguments**
- `flag`  *logical.*

**Value**
Previous value of the modified option.

---

**white_body.spct**  *Theoretical white body*

---

**Description**
A dataset for a hypothetical object with transmittance 0/1 (0%), reflectance 1/1 (100%)

**Format**
A object_spct object with 4 rows and 3 variables

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

**See Also**
**white_led.cps_spct**

**White led bulb spectrum**

**Description**

A dataset containing wavelengths and the corresponding spectral data as counts per second for an Osram warm white led lamp:

**Usage**

white_led.cps_spct

**Format**

A data.frame object with 2068 rows and 2 variables

**Details**

- w.length (nm), range 188 to 1117 nm.
- cps

**See Also**


**Examples**

white_led.cps_spct

---

**white_led.raw_spct**

**White led bulb spectrum**

**Description**

A dataset containing wavelengths and the corresponding spectral data as raw instrument counts for an Osram warm white led lamp, for three different integration times:

**Usage**

white_led.raw_spct
white_led.source_spct

Format
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 188 to 1117 nm.
- counts_1
- counts_2
- counts_3
- w.length (nm), range 188 to 1117 nm.
- cps

See Also

Examples
white_led.raw_spct

---

white_led.source_spct  White led bulb spectrum

Description
A dataset containing wavelengths and the corresponding spectral irradiance data for an Osram warm white led lamp:

Usage
white_led.source_spct

Format
A source_spct object with 1421 rows and 2 variables

Details
- w.length (nm), range 250 to 900 nm.
- s.e.irrad (W m^-2 nm^-1)
See Also


Examples

```r
white_led.source_spct
```

---

**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

```r
wls_at_target(
  x,
  target = NULL,
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)
```

```r
## Default S3 method:
wls_at_target(
  x,
  target = NULL,
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)
```

```r
## S3 method for class 'data.frame'
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
```

```r```
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  col.name = NULL,
  y.var.name = col.name,
  ...  
)

## S3 method for class 'source_spct'

wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...  
)

## S3 method for class 'response_spct'

wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...  
)

## S3 method for class 'filter_spct'

wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
...

## S3 method for class 'reflector_spct'
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)

## S3 method for class 'generic_mspct'
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
,
  .parallel = FALSE,
  .paropts = NULL
)

Arguments

x data.frame or spectrum object.
target numeric value indicating the spectral quantity value for which wavelengths are to be searched and interpolated if need. The character string "half.maximum" is also accepted as argument.
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 spct.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 or 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 wavelengths for the target value is returned.

Methods (by class)
- default: Default returning always an empty object of the same class as x.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct" objects.
- cps_spct: Method for "cps_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.

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

```r
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")
```


---

### Description

A dataset containing wavelengths at a 1 nm interval (100 nm to 4000 nm) and the corresponding CMF and CC colors for human vision.

### Usage

```r
wl_colors.spct
```

### Format

A `generic_spct` object with 3901 rows and 3 variables.

### Details

- `w.length (nm)`, numeric vector
- `CMF`, named character vector
- `CC`, named character vector

### Note

Data computed with function `color_of()` and used by function `fast_color_of_wl()`.

A `chroma_spct` object with variables as follows:

### Examples

```r
wl_colors.spct
```
**wl_max**

**Wavelength maximum**

**Description**

A method specialization that returns the wavelength maximum from objects of classes "waveband" or of class "generic_spct" or derived.

**Usage**

```r
wl_max(x, na.rm = FALSE)
```

```r
## S3 method for class 'waveband'
max(..., na.rm = FALSE)
```

```r
## S3 method for class 'generic_spct'
max(..., na.rm = FALSE)
```

```r
## 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
- **...**: not used in current version
- **idx**: character Name of the column with the names of the members of the collection of spectra.

**Methods (by class)**

- **generic_spct**:
- **generic_mspct**:

**Examples**

```r
max(sun.spct)
wl_max(sun.spct)
```
Description

A function that returns the wavelength (or value) at the center of the wavelength range of a waveband or spectrum object (or 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 \((\text{max}(x) - \text{min}(x)) / 2\). In the case of spectral objects a wavelength in nm. For any other R object, according to available definitions of \text{min} and \text{max}.

Methods (by class)

- default: Default method for generic function
- numeric: Default method for generic function
- waveband: Wavelength at center of a "waveband".
• `generic_spct`: Method for "generic_spct".
• `generic_mspct`: Method for "generic_mspct" objects.

See Also

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 from objects of classes "waveband" or of class "generic_spct" or derived.

**Usage**

```r
wl_min(x, na.rm = FALSE)
```

```r
## S3 method for class 'waveband'
min(..., na.rm = FALSE)
```

```r
## S3 method for class 'generic_spct'
min(..., na.rm = FALSE)
```

```r
## 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.
Methods (by class)

- generic_spct:
- generic_mspct:

See Also

Other wavelength summaries: `wl_midpoint()`, `wl_range()`, `wl_stepsize()`

Examples

\[
\text{min(sun.spct)} \\
\text{wl_min(sun.spct)}
\]

---

**wl_range**  
**Wavelength range**

Description

A method specialization that returns the wavelength range from objects of classes "waveband" or of class 'generic_spct' or derived.

Usage

\[
\text{wl_range(x, na.rm = FALSE)}
\]

\[
\text{\# S3 method for class 'waveband'} \\
\text{range(..., na.rm = FALSE)}
\]

\[
\text{\# S3 method for class 'generic_spct'} \\
\text{range(..., na.rm = FALSE)}
\]

\[
\text{\# S3 method for class 'generic_mspct'} \\
\text{range(..., na.rm = FALSE, idx = "spct.idx")}
\]

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.

Methods (by class)

- generic_spct:
- generic_mspct:
See Also

Other wavelength summaries: \texttt{wl_midpoint()}, \texttt{wl_min()}, \texttt{wl_stepsize()}

Examples

```
range(sun.spct)
w_range(sun.spct)
range(sun.spct)
```

##

<table>
<thead>
<tr>
<th>\texttt{wl_stepsize}</th>
<th>\textit{Stepsize}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Function that returns the range of step sizes in an object. Range of differences between successive sorted values.

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

- \texttt{x}: an R object
- \texttt{...}: not used in current version
- \texttt{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.
w_length2rgb

Methods (by class)

- default: Default function usable on numeric vectors.
- numeric: Method for numeric vectors.
- generic_spct: Method for "generic_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.

See Also

Other wavelength summaries: \texttt{wl_midpoint()}, \texttt{wl_min()}, \texttt{wl_range()}

Examples

\begin{verbatim}
stepsizesun.spct
wl_stepsizesun.spct
stepsizesun.spct
\end{verbatim}

\begin{longtable}{ll}
\textbf{w_length2rgb} & \textit{Wavelength to rgb color conversion} \\
\hline
\end{longtable}

Description

Calculates rgb values from spectra based on human color matching functions.

Usage

\begin{verbatim}
\texttt{w_length2rgb(w.length, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)}
\end{verbatim}

Arguments

\begin{verbatim}
w.length numeric Vector of wavelengths (nm) 
sens chroma_spct Used as chromaticity definition 
color.name character Used for naming the rgb color definition
\end{verbatim}

Value

A vector of colors defined using \texttt{rgb()}. The numeric values of the RGB components can be obtained using function \texttt{col2rgb()}. 

See Also

Other color functions: \texttt{rgb_spct()}, \texttt{w_length_range2rgb()}
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))
yellow_gel.spct  

| yellow_gel.spct | Transmittance spectrum of yellow theatrical gel. |

**Description**

A dataset containing the wavelengths at a 1 nm interval and fractional total transmittance for polyester film.

**Usage**

yellow_gel.spct

**Format**

A filter_spct object with 611 rows and 2 variables

**Details**

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

**See Also**


**Examples**

yellow_gel.spct

---

```
^ .generic_spct  Arithmetic Operators
```

**Description**

Power operator for spectra.

**Usage**

```r
# 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

*Topic **datasets**

A.illuminant.spct, 10
beesxyzCMF.spct, 52
black_body.spct, 53
ccd.spct, 56
ciev10.spct, 62
ciev2.spct, 63
ciexyzCC10.spct, 64
ciexyzCC2.spct, 65
ciexyzCMF10.spct, 66
ciexyzCMF2.spct, 67
clear.spct, 73
clear_body.spct, 74
cone_fundamentals10.spct, 79
D2.UV586, 86
D2.UV653, 86
D2.UV654, 87
D65.illuminant.spct, 88
FEL.BN.9101.165, 126
green_leaf.spct, 166
Ler_leaf.spct, 194
Ler_leaf.rfl.t.spct, 195
Ler_leaf.trns.spct, 196
Ler_leaf.trns_i.spct, 197
opaque.spct, 214
photodiode.spct, 222
polyester.spct, 228
sun.daily.data, 310
sun.daily.spct, 311
sun.data, 312
sun.spct, 313
white_body.spct, 374
white_led.cps_spct, 375
white_led.raw_spct, 375
white_led.source_spct, 376
wl_colors.spct, 381
yellow_gel.spct, 389

*Topic **internal**

v_insert_hinges, 363

v_replace_hinges, 364
*.generic_spct (times-.generic_spct), 338
+.generic_spct (plus-.generic_spct), 227
-.generic_spct (minus-.generic_spct), 201
/.generic_spct (slash-.generic_spct), 281
[.chroma_spct (Extract), 111
[.cps_spct (Extract), 111
[.filter_spct (Extract), 111
[.generic_mspct (Extract_mspct), 113
[.generic_spct (Extract), 111
[.object_spct (Extract), 111
[.raw_spct (Extract), 111
[.reflector_spct (Extract), 111
[.response_spct (Extract), 111
[.source_spct (Extract), 111
[<-.generic_mspct (Extract_mspct), 113
[<-.generic_spct (Extract), 111
[[<-.generic_mspct (Extract_mspct), 113
$<-.generic_mspct (Extract_mspct), 113
$<-.generic_spct (Extract), 111
%/_.generic_spct (div-.generic_spct), 101
%%.generic_spct (mod-.generic_spct), 202

A.illuminant.spct, 10, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
A2T, 11, 21, 22, 50, 104, 105, 234, 331, 333
A_as_default (energy_as_default), 106
abs.generic_spct (MathFun), 199
absorbance, 12, 212
absorptance, 15
acos.generic_spct (Trig), 341
add_attr2tb, 14, 17, 18, 110, 116, 119, 122, 124, 135, 177, 235, 238, 241, 244,
INDEX

as.source_spct, 25, 26, 29, 31, 34, 37, 40, 43, 45, 48, 288
as_quantum, 12, 21, 22, 49, 104, 105, 234, 331, 333
as_tod, 51
asin.generic_spct (Trig), 341
astrocalc4r, 91, 315
atan.generic_spct (Trig), 341
average_spct, 52
beesxyzCMF.spct, 52, 63–67, 80
black_body.spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
c, 54
calc_filter_multipliers (defunct), 92
calc_source_output, 55
calibration_mspct (generic_mspct), 146
calibration_spct (source_spct), 284
ceiling.generic_spct (round), 261
check_spct, 58, 61, 62, 105
check_spectrum, 61, 61, 62, 105, 299, 300, 302
check_w.length, 61, 62, 105
checkTimeUnit, 57, 81–83, 158, 275
cchroma_mspct (generic_mspct), 146
cchroma_spct (source_spct), 284
ciev10.spct, 53, 62, 64–67, 80
ciev2.spct, 53, 63, 63, 64–67, 80
ciexyzCC10.spct, 53, 63, 64, 64, 65–67, 80
ciexyzCC2.spct, 53, 63, 64, 65, 66, 67, 80
ciexyzCMF10.spct, 53, 63–65, 66, 67, 80
ciexyzCMF2.spct, 53, 63–66, 67, 80
class_spct, 68
clean, 68
clear_spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
clear_body_spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
clip_wl, 74, 345, 348, 350
col2rgb, 316, 388
color(color_of), 76
colour_of(color_of), 76
color(cps2rfr, 456, 83, 84, 280, 281,
compare_spct, 78
copy_attributes, 84
cor, 329
cos.generic_spct(Trig), 341
cps2irrad, 85
cps2rfr(cps2irrad), 85
cps2Tfr(cps2irrad), 85
cps_mspct(generic_mspct), 146
cps_spect(source_spect), 284
d2.uv586, 86
d2.uv653, 86
d2.uv654, 87
d2_spectrum, 87
d65.illuminant.spect, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
day_length(day_night), 89
day_night, 89, 136, 183, 230, 231, 283, 315
day_night_fast(day_night), 89
defunct, 92
despike, 93
diffraction_double_slit
(diffraction_single_slit), 99
diffraction_single_slit, 99
dim.generic_mspct, 100
dim<-.generic_mspct
(dim.generic_mspct), 100
disable_check_spct(enable_check_spct), 105
div_.generic_spct, 101
e2q, 12, 21, 22, 50, 103, 104, 105, 234, 331, 333
e2qmol_multipliers, 12, 21, 22, 50, 104, 105, 234, 331, 333
e2quantum_multiplier, 12, 21, 22, 50, 104, 105, 234, 331, 333
e_fluence, 114, 119, 135, 178, 239, 242
e_irrad, 116, 117, 135, 178, 239, 242
e_ratio, 111, 120, 236, 245
e_response, 123, 248, 257
enable_check_spct, 61, 62, 105
ergie_as_default, 106
eq_ratio, 109, 123, 236, 245
exp.generic_spct(log), 198
expanse(spread), 302
extend2extremes(trim_spct), 343
Extract, 111, 112, 113
Extract_mspct, 113
Extremes, 324
f_mspct(defunct), 92
fast_color_of_wl(color_of), 76
fast_color_of_wb(color_of), 76
fast_wb2rect_spect(wb2rect_spect), 371
fel_spec, 9101, 165, 126
fel_spect, 126
filter_cps.mspct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
filter_mspct(generic_mspct), 146
filter_properties
(getFilterProperties), 148
filter_properties<- 
  (setFilterProperties), 263
filter_spect (source_spect), 284
find_peaks, 127, 129, 165, 222, 254, 294, 
  363, 380
find_spikes, 99, 128, 128, 165, 222, 254, 
  294, 363, 380
find_wls, 130
findMultipleWl, 127
fit_peaks, 131
fit_valleys (fit_peaks), 131
generic_spect (round), 261
fluence, 116, 119, 133, 178, 239, 242
format, 137, 307
format.solar_time, 91, 136, 136, 183, 230, 
  231, 283, 315
format.tod_time, 91, 136, 136, 183, 230, 
  231, 283, 315
formatted_range, 137
fscale, 137, 146, 155, 156, 189, 190, 211, 
  272, 274
fshift, 142, 142, 155, 156, 189, 190, 211, 
  272, 274
generic_mspct, 146
generic_spect (source_spect), 284
geocode2tb (add_attr2tb), 18
get_attributes, 149, 151–153, 160–162, 
  163, 185, 186, 262, 264, 268, 270, 
  276, 278, 279, 289, 342, 343
get_peaks, 128, 129, 164, 222, 254, 294, 363, 
  380
get_spikes (get_peaks), 164
getAttType (defunct), 92
getBSWFUsed, 148, 263
getFilterProperties, 148, 151–153, 
  160–162, 164, 185, 186, 262, 264, 
  268, 270, 276, 278, 279, 289, 342, 
  343
getHowMeasured, 149, 150, 152, 153, 
  160–162, 164, 185, 186, 262, 264, 
  268, 270, 276, 278, 279, 289, 342, 
  343
getIdFactor, 151, 269
getInstrDesc, 149, 151, 152, 153, 160–162, 
  164, 185, 186, 262, 264, 268, 270, 
  276, 278, 279, 289, 342, 343
getInstrSettings, 149, 151, 152, 153, 
  160–162, 164, 185, 186, 262, 264, 
  268, 270, 276, 278, 279, 289, 342, 
  343
is.any_spct (is.generic_spct), 180
is.any_spect (is.generic_spct), 181
is.any_summary_spct
(is.summary_generic_spct), 183
is.calibration_spct
(is.generic_spct), 180
is.calibration_spct (is.generic_spct), 181
is.chroma_spct
(is.generic_spct), 181
is.cps_spct
(is.generic_spct), 180
is.cps_spct (is.generic_spct), 181
is.filter_spct
(is.generic_spct), 180
is.generic_spct
(is.generic_spct), 181
is.generic_spct (is.generic_spct), 180
is.object_spct
(is.generic_spct), 181
is.object_spct (is.generic_spct), 181
is.old_spct
(182, 354, 355)
is.raw_spct
(is.generic_spct), 180
is.raw_spct (is.generic_spct), 181
is.reflector_spct
(is.generic_spct), 180
is.reflector_spct (is.generic_spct), 181
is.response_spct
(is.generic_spct), 180
is.response_spct (is.generic_spct), 181
is.solar_date
(is.solar_time), 183
is.solar_time
(91, 136, 183, 230, 231, 283, 315)
is.source_spct
(is.generic_spct), 180
is.source_spct (is.generic_spct), 181
is.summary_chroma_spct
(is.summary_generic_spct), 183
is.summary_cps_spct
(is.summary_generic_spct), 183
is.summary_filter_spct
(is.summary_generic_spct), 183
is.summary_generic_spct
(is.summary_object_spct), 183
is.summary_object_spct
(is.summary_generic_spct), 183
is.summary_raw_spct
(is.summary_generic_spct), 183
is.summary_reflector_spct
(is.summary_generic_spct), 183
is.summary_response_spct
(is.summary_generic_spct), 183
is.summary_source_spct
(is.summary_generic_spct), 183
is.waveband, 184
is_absorbance_based, 186, 190
is_absorption_based
(is.absorbance_based), 186
is_effective, 187, 193, 207
is_energy_based(is.photon_based), 189
is_normalised(is.normalized), 188
is_normalized, 142, 146, 155, 156, 188, 190, 211, 272, 274
is_photon_based, 187, 189
is_scaled, 142, 146, 155, 156, 189, 190, 211, 272, 274
is_tagged, 191, 334, 353, 372, 373
is_transmittance_based
(is.absorbance_based), 186
is_valid_geocode(validate_geocode), 356
isValidInstrDesc, 149, 151–153, 160–162, 164, 185, 186, 262, 264, 268, 270, 276, 278, 279, 289, 342, 343
isValidInstrSettings, 149, 151–153, 160–162, 164, 185, 186, 262, 264, 268, 270, 276, 278, 279, 289, 342, 343
join, 200
join_mspct, 191
labels, 188, 193, 207
lat2tb (add_attr2tb), 18
length_geocode(validate_geocode), 356
Ler_leaf.rflt.spect, 11, 53, 57, 73, 74, 88, 166, 194, 195, 196, 197, 214, 223, 228, 310–313, 374–377, 389
Ler_leaf_trns.spect, 11, 53, 57, 73, 74, 88, 166, 194, 195, 196, 197, 214, 223, 228, 310–313, 374–377, 389
Ler_leaf_trns_i.spct, 11, 53, 57, 73, 74, 88, 166, 194–196, 197, 214, 223, 228, 310–313, 374–377, 389
log, 84, 101, 198, 199, 201, 202, 227, 261, 280, 281, 338, 390
log10.generic_spct (log), 198
log2.generic_spct (log), 198
lon2tb (add_attr2tb), 18
lonlat2tb (add_attr2tb), 18
mat2mspct (as.generic_mspct), 32
MathFun, 84, 101, 198, 199, 201, 202, 227, 261, 280, 281, 338, 390
max, 303, 383
max (wl_max), 382
mean, 318, 320
median, 321
merge2object_spct, 199
merge_attributes, 200
midpoint (wl_midpoint), 383
min, 303, 383
min (wl_min), 384
minus-.generic_spct, 201
mod-.generic_spct, 202
msaply (msmsply), 202
msdply (msmsply), 202
mslply (msmsply), 202
msmsply, 202
mspt2mat (as.matrix-mspct), 34
mspct_classes, 203
mutate_mspct (defunct), 92
NA, 137
na.action, 206
na.exclude.chroma_spct (na.omit), 204
na.exclude.cps_spct (na.omit), 204
na.exclude.filter_spct (na.omit), 204
na.exclude.generic_mspct (na.omit), 204
na.exclude.generic_spct (na.omit), 204
na.exclude.object_spct (na.omit), 204
na.exclude.raw_spct (na.omit), 204
na.exclude.reflector_spct (na.omit), 204
na.exclude.response_spct (na.omit), 204
na.exclude.source_spct (na.omit), 204
na.fail, 206
na.omit, 204
na_geocode (validate_geocode), 356
NDI (normalized_diff_ind), 211
new_waveband (waveband), 368
night_length (day_night), 89
noon_time (day_night), 89
normalise (normalize), 207
normalised_diff_ind
(normalized_diff_ind), 211
normalization, 188, 193, 206
normalize, 142, 146, 155, 156, 189, 190, 207, 272, 274
normalize_range_arg, 213
normalized_diff_ind, 211
object_mspct (generic_mspct), 146
object_spct (source_spct), 284
opaque_spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389
paste, 137
peaks, 128, 129, 165, 216, 254, 294, 363, 380
photobiology (photobiology-package), 8
photobiology-package, 8
photobiologyWavebands, 212
photodiode_spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 222, 228, 310–313, 374–377, 389
 photon_as_default (energy_as_default), 106
 plus-.generic_spct, 227
print, 228
print.solar_date (print.solar_time), 229
INDEX

print.solar_time, 91, 136, 183, 229, 231, 283, 315
print.summary.generic_spct, 230
print.tod_time, 91, 136, 183, 230, 231, 283, 315
print.waveband, 231
prod, 323
q2e, 12, 21, 22, 50, 104, 105, 233, 331, 333
q_fluence, 116, 119, 135, 178, 237, 242
q_irrad, 116, 119, 135, 178, 239, 239
q_ratio, 111, 123, 236, 242
q_response, 125, 245, 257
qe_ratio, 111, 123, 234, 245
quantum_as_default (energy_as_default), 106
range, 137
range (wl_range), 385
raw_mspct (generic_mspct), 146
raw_spct (source_spct), 284
rbindspect, 248
reflectance, 212, 250
reflector_mspct (generic_mspct), 146
reflector_spct (source_spct), 284
relative_AM, 252
replace_bad_pixs, 99, 128, 129, 165, 222, 253, 294, 363, 380
response, 125, 212, 248, 255
response_mspct (generic_mspct), 146
response_spct (source_spct), 284
Rfr_from_n, 257
Rfr_p_from_n (Rfr_from_n), 257
Rfr_s_from_n (Rfr_from_n), 257
rgb, 316
rgb_spct, 258, 387, 388
rmDerivedMspct, 259, 280
rmDerivedSpct, 260, 267
round, 84, 101, 198, 199, 201, 202, 227, 261, 280, 281, 338, 390
s_mean, 317
s_mean_se, 319
s_median, 320
s_prod, 321
s_range, 323
s_sd, 324
s_se, 325
s_sum, 327
s_var, 328
sd, 325
set_check_spct (enable_check_spct), 105
setAfrType (defunct), 92
setBSWFUsed, 148, 263
setCalibrationSpct (setGenericSpct), 265
setChromaSpct (setGenericSpct), 265
setCpsSpct (setGenericSpct), 265
setFilterSpct (setGenericSpct), 265
setGenericSpct, 25, 26, 29, 31, 34, 37, 40, 43, 45, 48, 260, 265
setIdFactor, 152, 269
setMultipleWL, 154, 271
setNormalised (setNormalized), 271
setNormalized, 142, 146, 155, 156, 189, 190, 211, 271, 274
setObjectSpct (setGenericSpct), 265
setRawSpct (setGenericSpct), 265
setReflectorSpct (setGenericSpct), 265
setRfrType, 156, 272
setScaled, 142, 146, 155, 156, 189, 190, 211, 272, 273
setSourceSpct(setGenericSpct), 265
setTfrType, 158, 274
setTimeUnit, 58, 81–83, 158, 275
shared_member_class, 260, 280
sign, 84, 101, 198, 199, 201, 202, 227, 261, 280, 281, 388, 390
signif.generic_spct (round), 261
sin.generic_spct (Trig), 341
slash_.generic_spct, 281
smooth_spct, 281
solar_time, 91, 136, 183, 230, 231, 283, 315
source_mspct (generic_mspct), 146
source_spct, 25, 26, 29, 31, 34, 37, 40, 43, 45, 48, 284
spct_classes (select_spct_attributes), 262
spct_classes, 289
spikes, 128, 129, 165, 222, 254, 289, 363, 380
splinefun, 173
split2calibration_mspct (split2mspc), 294
split2cps_mspct (split2mspc), 294
split2filter_mspct (split2mspc), 294
split2mspc, 24, 26, 28, 31, 33, 36, 39, 42, 44, 47, 294, 306
split2raw_mspct (split2mspc), 294
split2reflector_mspct (split2mspc), 294
split2response_mspct (split2mspc), 294
split2source_mspct (split2mspc), 294
split_bands, 297, 369
split_irradiance, 299
spread, 302
sqrt.generic_spct (MathFun), 199
stepsize (wl_stepsize), 386
strict_range_as_default
(VERBOSE AS_DEFAULT), 363
Subset, 304
subset, 113
subset.generic_spct (Subset), 304
subset2mspc, 24, 26, 28, 31, 33, 36, 39, 42, 44, 47, 297, 305
sum, 328
summary, 307
summary_spectra_classes, 308
sun.daily.data, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310, 311–313, 374–377, 389
sun.daily.spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310, 311, 312, 313, 374–377, 389
sun.data, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310, 311, 312, 313, 374–377, 389
sun_spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–312, 313, 374–377, 389
sun_angles, 91, 136, 183, 230, 231, 283, 314
sun_angles_fast (sun_angles), 314
sun_azimuth (sun_angles), 314
sun_elevation (sun_angles), 314
sun_zenith_angle (sun_angles), 314
sunrise_time (day_night), 89
sunset_time (day_night), 89
T2A, 12, 21, 22, 50, 104, 105, 234, 330, 333
T2Af, 12, 21, 22, 50, 104, 105, 234, 331, 331
T2T (defunct), 92
tag, 78, 191, 333, 353, 372, 373
tan.generic_spct (Trig), 341
wls_at_target, 128, 129, 165, 222, 254, 294, 363, 377

yellow_gel.spct, 11, 53, 57, 73, 74, 88, 166, 194–197, 214, 223, 228, 310–313, 374–377, 389