Package ‘photobiology’

August 5, 2022

Type Package

Title Photobiological Calculations

Version 0.10.12

Date 2022-07-21


License GPL (>= 2)

Depends R (>= 3.6.0)

Imports stats, grDevices, polynom (>= 1.4-1), tibble (>= 3.1.0), stringr (>= 1.4.0), lubridate (>= 1.7.8), plyr (>= 1.8.7), dplyr (>= 1.0.9), tidyr (>= 1.2.0), splus2R (>= 1.3-3), zoo (>= 1.8-8), rlang (>= 0.4.8)

Suggests knitr (>= 1.30), rmarkdown (>= 2.4), testthat (>= 2.3.2), roxygen2 (>= 7.1.1), lutz (>= 0.3.1)

LazyLoad yes

LazyData yes

ByteCompile true

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

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

Encoding UTF-8

RoxygenNote 7.2.1

VignetteBuilder knitr
NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (https://orcid.org/0000-0003-3385-972X),
Titta K. Koitilainen [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 2022-08-05 11:50:02 UTC

R topics documented:

- photobiology-package ................................................. 9
- A.illuminant.spect .................................................. 11
- A2T ................................................................. 12
- absorbance ......................................................... 13
- absorptance ......................................................... 16
- add_attr2tb ......................................................... 19
- Afr2T ............................................................... 21
- any2T ............................................................... 23
- as.calibration_mspct ............................................. 24
- as.calibration_spct ............................................... 26
- as.chroma_mspct ................................................... 27
- as.chroma_spct ..................................................... 28
- as.cps_mspct ......................................................... 28
- as.cps_spct ........................................................ 30
- as.filter_mspct ...................................................... 31
- as.filter_spct ........................................................ 33
- as.generic_mspct .................................................... 34
- as.generic_spct ..................................................... 36
- as.matrix_mspct ..................................................... 37
- as.object_mspct ..................................................... 38
- as.object_spct ....................................................... 39
- as.raw_mspct ........................................................ 40
- as.raw_spct ........................................................ 42
- as.reflector_mspct .................................................. 42
- as.reflector_spct .................................................... 44
- as.response_mspct .................................................. 45
- as.response_spct .................................................... 47
- as.solar_date ......................................................... 48
- as.solute_mspct ...................................................... 49
- as.solute_spct ........................................................ 51
- as.source_mspct ..................................................... 52
- as.source_spct ....................................................... 54
- as_energy ............................................................ 56
- as_quantum ........................................................... 56
- as_quantum_mol ...................................................... 57
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_tod</td>
<td>58</td>
</tr>
<tr>
<td>average_spct</td>
<td>59</td>
</tr>
<tr>
<td>beesxyzCMF.spct</td>
<td>59</td>
</tr>
<tr>
<td>black_body.spct</td>
<td>60</td>
</tr>
<tr>
<td>c</td>
<td>61</td>
</tr>
<tr>
<td>calc_multipliers</td>
<td>61</td>
</tr>
<tr>
<td>calc_source_output</td>
<td>62</td>
</tr>
<tr>
<td>ccd.spct</td>
<td>63</td>
</tr>
<tr>
<td>checkTimeUnit</td>
<td>64</td>
</tr>
<tr>
<td>check_spct</td>
<td>65</td>
</tr>
<tr>
<td>check_spectrum</td>
<td>68</td>
</tr>
<tr>
<td>check_w.length</td>
<td>69</td>
</tr>
<tr>
<td>ciev10.spct</td>
<td>70</td>
</tr>
<tr>
<td>ciev2.spect</td>
<td>71</td>
</tr>
<tr>
<td>cievxyzCC10.spct</td>
<td>72</td>
</tr>
<tr>
<td>cievxyzCC2.spect</td>
<td>73</td>
</tr>
<tr>
<td>cievxyzCMF10.spct</td>
<td>74</td>
</tr>
<tr>
<td>cievxyzCMF2.spect</td>
<td>75</td>
</tr>
<tr>
<td>class_spct</td>
<td>76</td>
</tr>
<tr>
<td>clean</td>
<td>76</td>
</tr>
<tr>
<td>clear.spct</td>
<td>81</td>
</tr>
<tr>
<td>clear_body.spct</td>
<td>82</td>
</tr>
<tr>
<td>clip_wl</td>
<td>83</td>
</tr>
<tr>
<td>collect2mspect</td>
<td>84</td>
</tr>
<tr>
<td>color_of</td>
<td>85</td>
</tr>
<tr>
<td>compare_spct</td>
<td>87</td>
</tr>
<tr>
<td>cone_fundamentals10.spct</td>
<td>89</td>
</tr>
<tr>
<td>convertTfrType</td>
<td>90</td>
</tr>
<tr>
<td>convertThickness</td>
<td>91</td>
</tr>
<tr>
<td>convertTimeUnit</td>
<td>92</td>
</tr>
<tr>
<td>convolve_each</td>
<td>93</td>
</tr>
<tr>
<td>copy_attributes</td>
<td>94</td>
</tr>
<tr>
<td>cps2irrad</td>
<td>95</td>
</tr>
<tr>
<td>D2.UV586</td>
<td>96</td>
</tr>
<tr>
<td>D2.UV653.</td>
<td>96</td>
</tr>
<tr>
<td>D2.UV654.</td>
<td>97</td>
</tr>
<tr>
<td>D2_spectrum</td>
<td>97</td>
</tr>
<tr>
<td>D65.illuminant.spct</td>
<td>98</td>
</tr>
<tr>
<td>day_night</td>
<td>99</td>
</tr>
<tr>
<td>defunct</td>
<td>102</td>
</tr>
<tr>
<td>despike</td>
<td>103</td>
</tr>
<tr>
<td>diffraction_single_slit</td>
<td>110</td>
</tr>
<tr>
<td>dim.generic_mspct</td>
<td>111</td>
</tr>
<tr>
<td>div-.generic_spct</td>
<td>111</td>
</tr>
<tr>
<td>div_spectra</td>
<td>112</td>
</tr>
<tr>
<td>drop_user_cols</td>
<td>113</td>
</tr>
<tr>
<td>e2q</td>
<td>115</td>
</tr>
<tr>
<td>e2qmol_multipliers</td>
<td>116</td>
</tr>
</tbody>
</table>
R topics documented:

- e2quantum_multipliers ........................................... 117
- enable_check_spct .................................................. 117
- energy_as_default .................................................. 118
- energy_irradiance ............................................... 119
- energy_ratio ........................................................ 120
- eq_ratio ............................................................. 121
- ET_ref ............................................................... 123
- Extract ............................................................. 126
- Extract_mspct ..................................................... 128
- e_fluence ........................................................... 129
- e_irrad ............................................................... 132
- e_ratio ............................................................... 135
- e_response .......................................................... 138
- FEL.BN.9101.165 .................................................. 141
- FEL_spectrum ....................................................... 141
- findMultipleWl ..................................................... 142
- find_peaks .......................................................... 142
- find_spikes ........................................................ 143
- find_wls .............................................................. 145
- fit_peaks ............................................................ 146
- fluence .............................................................. 148
- format.solar_time ................................................ 151
- format.tod_time ................................................... 151
- formatted_range ................................................... 152
- fscale ............................................................... 152
- fshift ............................................................... 158
- generic_mspct ..................................................... 163
- getBSWFUsed ....................................................... 164
- getFilterProperties .............................................. 165
- getHowMeasured .................................................. 166
- getIdFactor ........................................................ 168
- getInstrDesc ....................................................... 168
- getInstrSettings .................................................. 169
- getKType ............................................................ 170
- getMspctVersion .................................................. 170
- getMultipleWl ..................................................... 171
- getNormalized ..................................................... 172
- getResponseType .................................................. 173
- getRfrType ........................................................ 174
- getScaled .......................................................... 174
- getSoluteProperties ............................................. 175
- getSpctVersion ..................................................... 177
- getTfrType ........................................................ 177
- getTimeUnit ......................................................... 178
- getWhatMeasured .................................................. 179
- getWhenMeasured .................................................. 180
- getWhereMeasured ................................................ 181
- get_attributes ..................................................... 183
R topics documented:

get_peaks .................................................. 184
green_leaf.spct ........................................... 186
head_tail .................................................... 187
insert_hinges ............................................. 188
insert_spect_hinges ...................................... 189
integrate_spect .......................................... 190
integrate_xy .............................................. 191
interpolate_spect ........................................ 192
interpolate_spectrum .................................... 193
interpolate_wl ........................................... 194
irrad ....................................................... 196
irradiance ............................................... 199
irrad_extraterrestrial .................................. 200
is.generic_mspct ........................................ 201
is.generic_spect ......................................... 203
is.old_spect ............................................. 204
is.solar_time ........................................... 204
is.summary_generic_spect ............................... 205
is.waveband ............................................. 206
isValidInstrDesc ........................................ 206
isValidInstrSettings ................................... 207
is_absorbance_based ..................................... 208
is_effective ............................................. 209
is_mole_based ........................................... 210
is_normalized ............................................ 211
is_photon_based ......................................... 211
is_scaled .................................................. 212
is_tagged .................................................. 213
join_mspct ............................................... 214
labels ...................................................... 216
Ler_leaf.spct ............................................ 216
Ler_leaf_rflt.spct ....................................... 217
Ler_leaf_trns.spct ....................................... 218
Ler_leaf_trns_i.spct .................................... 219
log ......................................................... 220
MathFun .................................................... 221
merge2object_spect ...................................... 222
merge_attributes ......................................... 223
minus_generic_spect ..................................... 224
mod_generic_spect ....................................... 224
msmsply .................................................... 225
mspct_classes ............................................ 226
na.omit ..................................................... 226
net_irradiance ........................................... 229
normalization ............................................. 230
normalize .................................................. 231
normalized_diff_ind .................................... 237
normalize_range_arg .................................... 238
opaque.spct ............................ 239
oper_spectra ................................ 240
peaks .................................. 241
phenylalanine.spct ..................... 248
photodiode.spct ........................ 249
photons_energy_ratio .................. 250
photon_irradiance ...................... 251
photon_ratio ................................ 253
plus_.generic_spct ..................... 254
polyester.spct .......................... 255
print ................................... 255
print.solar_time ......................... 256
print.summary_generic_spect ............ 257
print.tod_time .......................... 258
print.waveband .......................... 258
prod_spectra ................................ 259
q2e .................................... 260
qe_ratio ................................ 261
q_fluence ................................ 264
q_irrad ................................ 266
q_ratio ................................ 270
q_response ................................ 272
r4p_pkgs ................................ 275
rbindspct ................................ 276
reflectance ................................ 277
relative_AM ................................ 280
replace_bad_pixs ........................ 281
response .................................. 283
Rfr_from_n ................................ 285
rgb_spct ................................ 286
rmDerivedMspct .......................... 287
rmDerivedSpct .......................... 288
round .................................... 289
select_spect_attributes ................. 290
setBSWFUsed ................................ 291
setFilterProperties ................. 292
setGenericSpct .......................... 293
setHowMeasured .......................... 297
setIdFactor ................................ 298
setInstrDesc ............................ 299
setInstrSettings ......................... 300
setKType ................................ 301
setMultipleWl ........................... 301
setNormalized .......................... 302
setResponseType ......................... 303
setRfrType ................................ 304
setScaled ................................ 305
setSoluteProperties ..................... 306
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>setTfrType</td>
<td>308</td>
</tr>
<tr>
<td>setTimeUnit</td>
<td>309</td>
</tr>
<tr>
<td>setWhatMeasured</td>
<td>310</td>
</tr>
<tr>
<td>setWhenMeasured</td>
<td>311</td>
</tr>
<tr>
<td>setWhereMeasured</td>
<td>312</td>
</tr>
<tr>
<td>shared_member_class</td>
<td>314</td>
</tr>
<tr>
<td>sign</td>
<td>314</td>
</tr>
<tr>
<td>slash-generic_spct</td>
<td>315</td>
</tr>
<tr>
<td>smooth_spct</td>
<td>315</td>
</tr>
<tr>
<td>solar_time</td>
<td>318</td>
</tr>
<tr>
<td>source_spct</td>
<td>319</td>
</tr>
<tr>
<td>spct_attr2tb</td>
<td>325</td>
</tr>
<tr>
<td>spct_classes</td>
<td>326</td>
</tr>
<tr>
<td>spct_metadata</td>
<td>326</td>
</tr>
<tr>
<td>spikes</td>
<td>328</td>
</tr>
<tr>
<td>split2mspct</td>
<td>333</td>
</tr>
<tr>
<td>split_bands</td>
<td>335</td>
</tr>
<tr>
<td>split_energy_irradiance</td>
<td>337</td>
</tr>
<tr>
<td>split_irradiance</td>
<td>338</td>
</tr>
<tr>
<td>split_photon_irradiance</td>
<td>340</td>
</tr>
<tr>
<td>spread</td>
<td>341</td>
</tr>
<tr>
<td>Subset</td>
<td>342</td>
</tr>
<tr>
<td>subset2mspct</td>
<td>343</td>
</tr>
<tr>
<td>subb_spectra</td>
<td>344</td>
</tr>
<tr>
<td>summary</td>
<td>346</td>
</tr>
<tr>
<td>summary_spct_classes</td>
<td>347</td>
</tr>
<tr>
<td>sum_spectra</td>
<td>347</td>
</tr>
<tr>
<td>sun.daily.data</td>
<td>348</td>
</tr>
<tr>
<td>sun.daily.spect</td>
<td>349</td>
</tr>
<tr>
<td>sun.data</td>
<td>350</td>
</tr>
<tr>
<td>sun.spect</td>
<td>351</td>
</tr>
<tr>
<td>sun_angles</td>
<td>352</td>
</tr>
<tr>
<td>s_e_irrad2rgb</td>
<td>355</td>
</tr>
<tr>
<td>s_mean</td>
<td>356</td>
</tr>
<tr>
<td>s_mean_se</td>
<td>358</td>
</tr>
<tr>
<td>s_mean_se_band</td>
<td>359</td>
</tr>
<tr>
<td>s_median</td>
<td>361</td>
</tr>
<tr>
<td>s_prod</td>
<td>362</td>
</tr>
<tr>
<td>s_range</td>
<td>364</td>
</tr>
<tr>
<td>s_sd</td>
<td>366</td>
</tr>
<tr>
<td>s_se</td>
<td>367</td>
</tr>
<tr>
<td>s_sum</td>
<td>369</td>
</tr>
<tr>
<td>s_var</td>
<td>371</td>
</tr>
<tr>
<td>T2A</td>
<td>372</td>
</tr>
<tr>
<td>T2Afr</td>
<td>374</td>
</tr>
<tr>
<td>tag</td>
<td>376</td>
</tr>
<tr>
<td>thin_wl</td>
<td>378</td>
</tr>
<tr>
<td>times-generic_spct</td>
<td>381</td>
</tr>
</tbody>
</table>
transmittance ............................................................ 381
Trig ................................................................. 384
trimInstrDesc ........................................................... 385
trimInstrSettings .......................................................... 386
trim_spect .............................................................. 387
trim_tails ............................................................... 389
trim_waveband .......................................................... 390
trim_wl ................................................................. 392
tz_time_diff ............................................................ 394
uncollect2spct .......................................................... 395
untag ................................................................. 396
upgrade_spect ............................................................ 397
upgrade_spectra .......................................................... 398
using_Tfr ............................................................... 398
validate_geocode .......................................................... 399
valleys ................................................................. 400
verbose_as_default ......................................................... 407
v_insert_hinges .......................................................... 408
v_replace_hinges .......................................................... 408
water_spect ............................................................. 409
water_vp_sat ............................................................. 410
waveband ............................................................... 414
waveband_ratio ........................................................... 416
wb2rect_spect ............................................................ 417
wb2spct ................................................................. 418
wb2tagged_spect .......................................................... 419
wb_trim_as_default ......................................................... 420
white_body_spect .......................................................... 420
white_led_cps_spect ....................................................... 421
white_led_raw_spect ....................................................... 421
white_led_source_spect .................................................... 422
wls_at_target ............................................................ 423
wl_max ................................................................. 427
wl_midpoint .............................................................. 428
wl_min ................................................................. 430
wl_range ............................................................... 431
wl_stepsize ............................................................... 432
w_length2rgb .............................................................. 433
w_length_range2rgb ......................................................... 434
yellow_gel_spect .......................................................... 435
^generic_spect ............................................................ 435

Index ................................................................. 437
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 R packages supporting computations and plotting relevant to photobiology (described at https://www.r4photobiology.info/). Package ‘photobiology’ has its main focus in the characterization of the light environment, the description of optical properties of objects and substances and description of light responses of organisms and devices used to measure light. The facilities for spectral data storage and manipulations are widely useful in photobiology, chemistry, geophysics, radiation climatology and remote sensing. Astronomical computations for the sun are also implemented. The design of object classes for spectral data supports reproducibility by facilitating the consistent use of units and physical quantities and consistent embedding of metadata. Data are expressed throughout using SI base units, except for wavelengths which are consistently expressed in nanometres [nm]. Please see the vignette 0: The R for Photobiology Suite for a description of the suite.

Acknowledgements

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

Author(s)

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

Other contributors:

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


See Also

Useful links:

- [https://docs.r4photobiology.info/photobiology/](https://docs.r4photobiology.info/photobiology/)
- [https://github.com/aphalo/photobiology](https://github.com/aphalo/photobiology)

Examples

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

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

See Also

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

Description

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

Usage

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

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

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

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

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

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

Arguments

spct an R object.

w.band waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.

quantity character string One of "average", "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".

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

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

... other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx character Name of the column with the names of the members of the collection of spectra.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
absorbance

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

Value

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

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

Methods (by class)

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

Note

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

Examples

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

**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**

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

## Default S3 method:

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

## S3 method for class 'filter_spct'

```r
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'

```r
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'

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

attr2tb = NULL,
absorptance

```r
idx = "spct.idx"
)

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

### Arguments

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

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

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

Methods (by class)

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

Note

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

Examples

```r
absorptance(black_body.spct, new_waveband(400,500))
absorptance(white_body.spct, new_waveband(300,400))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "average")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "total")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "relative")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "relative.pc")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "contribution")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
            quantity = "contribution.pc")
```
add_attr2tb

Copy attributes from members of a generic_mspct

Description

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

Usage

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

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

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

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

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

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

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

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

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

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

scaled2tb(mspct, tb = NULL, col.names = "scaled", idx = "spct.idx")
add_attr2tb

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

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

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

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

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

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

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

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

Arguments

\begin{itemize}
\item \texttt{tb} \hspace{3cm} tibble or data.frame to which to add the data (optional).
\item \texttt{mspct} \hspace{3cm} \texttt{generic_mspct} Any collection of spectra.
\item \texttt{col.names} \hspace{3cm} named character vector Name(s) of metadata attributes to copy, while if named, the names provide the name for the column.
\item \texttt{idx} \hspace{3cm} character Name of the column with the names of the members of the collection of spectra.
\item \texttt{unnest} \hspace{3cm} logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.
\end{itemize}

Details

The attributes are copied to a column in a tibble or data frame. If the \texttt{tb} formal parameter receives \texttt{NULL} as argument, a new tibble will be created. If an existing \texttt{data.frame} or \texttt{tibble} is passed as argument, new columns are added to it. However, the number of rows in the argument passed to \texttt{tb} must match the number of spectra in the argument passed to \texttt{mspct}. Only in the case of method \texttt{add_attr2tb()} if the argument to \texttt{col.names} is a named vector, the names of members are used as names for the columns created. This permits setting any valid name for the new columns. If the vector passed to \texttt{col.names} has no names the names of the attributes are used for the new columns. If the fields of the attributes are unnested their names are used as names for the columns.
Valid accepted as argument to col.names are NULL, "lon", "lat", "address", "geocode", "where.measured", "when.measured", "what.measured", "how.measured", "comment", "normalised", "normalized", "scaled", "bswf.used", "instr.desc", "instr.settings", "filter.properties", "Tfr.type", "Rfr.type", "time.unit".

Value

A tibble with the metadata attributes in separate new variables.

Note

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.

See Also


Examples

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
- **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
any2T

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

Value

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

Methods (by class)

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

See Also

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

Examples

T2Afr(Ler_leaf.spct)

---

any2T

Convert filter quantities.

Description

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

Usage

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

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

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

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

Details

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

Value

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

See Also

Other quantity conversion functions: A2T(), Afr2T(), T2Afr(), T2A(), 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'

```r
as.calibration_mspct(x, ..., ncol = 1, byrow = FALSE)
```

## S3 method for class 'matrix'

```r
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)

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

### Note

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

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

as.calibration_spct

Coerce to a spectrum

Description

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

Usage

as.calibration_spct(x, ...)

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

Arguments

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

Value

A copy of x converted into a calibration_spct object.

Methods (by class)

• as.calibration_spct(default):

See Also

setGenericSpct

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

Coerce to a collection-of-spectra

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

Usage
as.chroma_mspct(x, ...)

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

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

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

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

Arguments
x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
...

... passed to individual spectrum object constructor
ncol integer Number of 'virtual' columns in data
byrow logical If ncol > 1 how to read in the data

Value
A copy of x converted into a chroma_mspct object.

Methods (by class)
- as.chroma_mspct(default):
- as.chroma_mspct(data.frame):
- as.chroma_mspct(chroma_spct):
- as.chroma_mspct(list):

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

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

Usage  
as.chroma_spct(x, ...)

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

Arguments  
x  
an R object.

...  
other arguments passed to "set" functions.

Value  
A copy of x converted into a chroma_spct object.

Methods (by class)  
• as.chroma_spct(default):

See Also  
setGenericSpct  
Other constructors of spectral objects: as.calibration_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.solute_spct(), as.source_spct(), source_spct()
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 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 cps_mspct object.
Methods (by class)

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

Note

When \( x \) is a square matrix an explicit argument is needed for \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.filter_mspct()}, \texttt{as.generic_mspct()}, \texttt{as.object_mspct()}, \texttt{as.raw_mspct()}, \texttt{as.reflector_mspct()}, \texttt{as.response_mspct()}, \texttt{as.solute_mspct()}, \texttt{as.source_mspct()}, \texttt{split2mspct()}, \texttt{subset2mspct()}

---

\begin{verbatim}
as.cps_spct

Coerce to a spectrum

Description

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

Usage

\texttt{as.cps_spct(x, \ldots)}

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

Arguments

\begin{itemize}
  \item \texttt{x} \quad \text{an R object.}
  \item \ldots \quad \text{other arguments passed to "set" functions.}
\end{itemize}

Value

A copy of \( x \) converted into a \texttt{cps_spct} object.

Methods (by class)

• as.cps_spct(default):
See Also

setGenericSpct

Other constructors of spectral objects:

- `as.calibration_spct()
- as.chroma_spct()
- as.filter_spct()
- as.generic_spct()
- as.object_spct()
- as.raw_spct()
- as.reflector_spct()
- as.response_spct()
- as.solute_spct()
- as.source_spct()

---

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

**Description**

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

**Usage**

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

## Default S3 method:

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

## S3 method for class 'data.frame'

```r
as.filter_mspct(x, Tfr.type = c("total", "internal"), strict.range = TRUE, ...)
```

## S3 method for class 'filter_spct'

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

## S3 method for class 'list'

```r
as.filter_mspct(  
  x,  
  Tfr.type = c("total", "internal"),  
  strict.range = TRUE,  
  ...  
  ncol = 1,  
  byrow = FALSE  
)
```

## S3 method for class 'matrix'

```r
as.filter_mspct(  
  x,  
  w.length,  
  spct.data.var = "Tfr",  
  multiplier = 1,  
  byrow = NULL,  
  spct.names = "spct_",  
  ...
)
```
Arguments

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

Value

A copy of x converted into a filter_mspct object.

Methods (by class)

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

Note

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

See Also

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

Coerce or convert into a filter spectrum

Description

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

Usage

as.filter_spect(x, ...)

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

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

Arguments

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

numeric Concentration to be used to compute transmittance of the solute in solution [mol m^{-3} = mmol dm^{-3} or kg m^{-3} = g dm^{-3}, respectively].

path.length numeric The length of the light path (m) used to compute transmittance of the solute in a solution.

Value

A copy of x converted into a filter_spct object.

Methods (by class)

• as.filter_spct(default):
• as.filter_spct(solute_spct):

See Also

setGenericSpct

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

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( 
as.generic_mspct

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

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

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
...
 passed to individual spectrum object constructor
force.spct.class logical indicating whether to change the class of members to generic_spct or retain the existing class.
ncol integer Number of ‘virtual’ columns in data
byrow logical If ncol > 1 how to read in the data
w.length numeric A vector of 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)

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

Note

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

See Also

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

Description

Coerce to a spectrum

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

Usage

as.generic_spct(x, ...)

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

Arguments

x an R object

... other arguments passed to "set" functions

Value

A copy of x converted into a generic_spct object.

Methods (by class)

- as.generic_spct(default):
as.matrix-mspct

See Also

setGenericSpct

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

---

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

Description

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

Usage

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

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

Arguments

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

Warning!

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

Note

Only collections of spectra containing spectra with exactly the same `w.length` values can 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,
  ...
)

## 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.
...
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 an object_mspct object.

Methods (by class)

- \texttt{as.object_mspct}(default):
- \texttt{as.object_mspct}(data.frame):
- \texttt{as.object_mspct}(object_spct):
- \texttt{as.object_mspct}(list):

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.raw_mspct()}, \texttt{as.reflector_mspct()}, \texttt{as.response_mspct()}, \texttt{as.solute_mspct()}, \texttt{as.source_mspct()}, \texttt{split\_2mspct()}, \texttt{subset\_2mspct()}

---

\texttt{as.object_spct} \hspace{1cm} \textit{Coerce to a spectrum}

---

Description

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

Usage

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

Arguments

- \( x \) \hspace{1cm} an R object.
- \( \ldots \) \hspace{1cm} other arguments passed to "set" functions.
- \( \text{Tfr.type} \) \hspace{1cm} a character string, either "total" or "internal".
- \( \text{Rfr.type} \) \hspace{1cm} a character string, either "total" or "specular".
- \( \text{strict.range} \) \hspace{1cm} logical Flag indicating whether off-range values result in an error instead of a warning.
Value

A copy of x converted into a object_spct object.

Methods (by class)

• as.object_spct(default):

See Also

setGenericSpct

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

---

as.raw_mspct  
Coerce to a collection-of-spectra

Description

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

Usage

as.raw_mspct(x, ...)

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

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

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

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

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

x  a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

ncol  integer Number of 'virtual' columns in data

byrow  logical If ncol > 1 how to read in the data

w.length  numeric A vector of 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 raw_mspct object.

Methods (by class)

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

Note

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

See Also

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

Coerce to a spectrum

Description

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

Usage

as.raw_spct(x, ...)

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

Arguments

x

an R object.

...

t other arguments passed to "set" functions.

Value

A copy of x converted into a raw_spct object.

Methods (by class)

• as.raw_spct(default):

See Also

setGenericSpct

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

Usage

as.reflector_mspct(x, ...)

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

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

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

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

## 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.
...
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
as.reflector_spct

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)

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

Note

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

See Also

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

as.reflector_spct Coerce to a spectrum

Description

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

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

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

Arguments

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

Value

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

Methods (by class)

- `as.reflector_spct` (default):

See Also

- `setGenericSpct`

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

---

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

Description

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

as.response_mspct(x, 

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

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

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

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

## S3 method for class 'matrix'
as.response_mspct(

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

time.unit character A string, "second", "day" or "exposure"

ncol integer Number of 'virtual' columns in data

byrow logical If ncol > 1 how to read in the data

w.length numeric A vector of 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)

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

Note

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

See Also

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

Description

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

Usage

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

## Default S3 method:

```r
as.response_spct(x, time.unit = "second", ...)
```

Arguments

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

Value

A copy of \( x \) converted into a `response_spct` object.
as.solar_date

Methods (by class)

• as.response_spct(default):

See Also

setGenericSpct

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

as.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.

See Also

Other Local solar time functions: is.solar_time(), print.solar_time(), solar_time()
as.solute_mspct

Coerce to a collection-of-spectra

Description

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

Usage

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

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

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

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

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

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

x  a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

K.type  a character string, either "attenuation", "absorption" or "scattering"

strict.range  logical Flag indicating how off-range values are handled

ncol  integer Number of 'virtual' columns in data

byrow  logical If ncol > 1 how to read in the data

w.length  numeric A vector of wavelength values sorted in strictly ascending order (nm).

spct.data.var  character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.

multiplier  numeric A multiplier to be applied to the values in x to do unit or scale conversion.

spct.names  character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a filter_mspct object.

Methods (by class)

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

Note

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

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspc(), subset2mspc()
as.solute_spct

Coerce to a solute spectrum

Description

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

Usage

as.solute_spct(x, ...)

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

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

Arguments

x

an R object.

... other arguments passed to "set" functions.

K.type a character string, one of "attenuation", "absorption" or "scattering". 
strict.range logical Flag indicating whether off-range values result in an error instead of a warning.

name, solvent.name character The names of the substance and of the solvent. A named character vector, with member names such as "IUPAC" for the authority.

mass numeric The mass in Dalton (Da = g/mol).

formula character The molecular formula.

structure raster A bitmap of the structure.

ID, solvent.ID character The IDs of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChen" for the authority.

comment character A string to be added as a comment attribute to the object created. If not supplied, the comment will be copied from x.

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

path.length numeric The length of the light path \((m)\) used to compute transmittance of the solute in a solution.

Value
A copy of x converted into a solute_spct object.

Methods (by class)
- as.solute_spct(default):
- as.solute_spct(filter_spct):

See Also
- setSoluteSpct

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

---

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:
as.source_mspct(x, ...)

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

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

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

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

Arguments

- **x**: a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
- **...**: passed to individual spectrum object constructor
- **time.unit**: character A string, "second", "day" or "exposure"
- **bswf.used**: character
Coerce to a spectrum

Description

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

as.source_spct(x, ...)

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

Arguments

x
  an R object.

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

Value

A copy of x converted into a source_spct object.

Methods (by class)

• as.source_spct(default):

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.solute_spct(), source_spct()
as_energy  
Convert spectral photon irradiance into spectral energy irradiance

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

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

Arguments
- w.length: numeric vector of wavelengths [\text{nm}].
- s.qmol.irrad: numeric vector of spectral photon irradiance values.

Value
A numeric vector of spectral (energy) irradiances.

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

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

as_quantum  
Convert spectral energy irradiance into spectral photon irradiance

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

Usage
as_quantum(w.length, s.e.irrad)
as_quantum_mol

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))
```
**Examples**

```r
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.

**See Also**

`solar_time`

Other Time of day functions: `format.tod_time()`, `print.tod_time()`

**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

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

Usage
average_spct(spct)

Arguments
spct generic_spct

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

Examples
average_spct(sun.spct)

beesxyzCMF.spct

Description
Honeybee xyz chromaticity colour matching function data

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

Usage
beesxyzCMF.spct

Format
A data frame with 81 rows and 4 variables
**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**

### Combine collections of spectra

**Description**

Combine two or more `generic_mspct` objects into a single object.

**Usage**

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

**Arguments**

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

**Value**

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

### calc_multipliers

**Spectral weights**

**Description**

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

**Usage**

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

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

Value

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

See Also

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

Examples

with(sun.data, calc_multipliers(w.length, new_waveband(400,700),"photon"))
with(sun.data, calc_multipliers(w.length, new_waveband(400,700),"photon"), 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)
)
```

ccd.spct

Spectral response of a back-thinned CCD image sensor.

Description

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

---

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

Description
Function to read the "time.unit" attribute

Usage
checkTimeUnit(x)

Arguments

x | a source_spct object
**check_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(), setTimeUnit()

---

**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(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

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

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

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

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

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

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

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

### Arguments

- **x**: An R object
- **byref**: logical indicating if new object will be created by reference or by copy of x
- **strict.range**: logical indicating whether off-range values result in an error instead of a warning, NA disables the test.
- **force**: logical If TRUE check is done even if checks are disabled.
... additional param possible in derived methods
multiple.wl numeric Maximum number of repeated w.length entries with same value.

Methods (by class)

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

See Also

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

Examples

```r
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)`
check_w.length

Arguments

w.length numeric vector of wavelengths \([nm]\).

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

Value

A single logical value indicating whether test was passed or not

See Also

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

Examples

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

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

- w.length (nm)
- 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
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

See Also


Examples

ciev2.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. Derived from proposed CIE 2006 standard. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

- \textit{w.length} (nm)
- \textit{x}
- \textit{y}
- \textit{z}

Usage

\texttt{ciexyzCC10.spct}

Format

A \texttt{chroma_spct} object with 441 rows and 4 variables

Author(s)

CIE

See Also

Other Visual response data examples: \texttt{beesxyzCMF.spct, ciev10.spct, ciev2.spct, ciexyzCC2.spct, ciexyzCMF10.spct, ciexyzCMF2.spct, cone_fundamentals10.spct}

Examples

\texttt{ciexyzCC10.spct}
ciexyzCC2.spct  CIE xyz chromaticity coordinates 2 deg data

Description

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

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

Usage

ciexyzCC2.spct

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also


Examples

ciexyzCC2.spct
ciexyzCMF10.spct

Linear energy CIE xyz colour matching function (CMF) 10 deg data

Description

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

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

Usage

ciexyzCMF10.spct

Format

A chroma_spct object with 441 rows and 4 variables

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
Extract class information from a generic spectrum.

Usage
class_spct(x)

Arguments
x
any R object

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

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

Examples
class_spct(sun.spct)
class(sun.spct)

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

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

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

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

## S3 method for class 'source_spct'
clean(
clean

x,
range = x,
range.s.data = c(0, NA),
fill = range.s.data,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...
)

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

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

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

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

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

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

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

## S3 method for class 'generic_spect'
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,
clean

...,
.parallel = FALSE,
.paropts = NULL
)

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

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

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

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

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

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

Arguments

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

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

Methods (by class)

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

Note

In the case of object_spct objects, cleaning is done first on the Rfr and Tfr columns and subsequently Afr estimated and if needed half of deviation of Afr from the expected minimum value subtracted from each of Rfr and Tfr.
clear_body.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_body.spct,
filter_cps.mspct,green_leaf.spct.opaque.spct,phenylalanine.spct,photodiode.spct,
polyester.spct,sun.daily.data,sun.daily.spct,sun.data,sun.spct,water.spct,white_body.spct,
white_led.cps_spct,white_led.raw_spct,white_led.source_spct,yellow_gel.spct

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
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,filter_cps.mspct,
green_leaf.spct.opaque.spct,phenylalanine.spct,photodiode.spct,polyester.spct,
sun.daily.data,sun.daily.spct,sun.data,sun.spct,water.spct,white_body.spct,white_led.cps_spct,
white_led.raw_spct,white_led.source_spct,yellow_gel.spct
**clip_wl**  
*Clip head and/or tail of a spectrum*

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

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

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

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

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

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

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

See Also

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

Examples

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

Description

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

Usage

```r
collect2mspect(
  .list = NULL,
  pattern = "*\.spct$",
  collection.class = NULL,
  ...)
```

Arguments

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

Details

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

Value

By default a collection of spectra.

See Also

Other experimental utility functions: drop_user_cols(), thin wl(), uncollect2spct()

Examples

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

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

-----------

color_of

Color of an object

Description

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

Usage

color_of(x, ...)

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

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


## Arguments

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

## Value

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

## Methods (by class)

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

Deprecated

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

Note

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 vectorized call to fast_color_of_wl() with a vector of waveband midpoint wavelengths.

Examples

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

Description

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

Usage

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

Arguments

x A collection of two spectral objects of the same type.
w.band waveband object or a numeric stepsize in nanometres.
.summary.fun function. The summary function to use. It must be a method accepting object x as first argument.

... additional named arguments passed down to .summary.fun.
.comparison.fun function. The comparison function to use.
returned.value character One of "data.frame", "spectrum", "tagged.spectrum".
use.hinges logical Flag indicating whether to insert "hinges" into the returned spectrum when tagging it.
short.names logical Flag indicating whether to use short or long names for wavebands when tagging.

Details

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

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

Value

A generic_spct, tagged or not with the 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))))
```
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

```r
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)
```
Value

A chroma_spct object.
A response_mspct object containing the same data in three response_spct objects.

Author(s)

CIE

See Also


Examples

cone_fundamentals10.spct

correctTfrType

Convert the "Tfr.type" attribute

Description

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

Usage

correctTfrType(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, a filter_spct object that is "under-the-hood" an object_spct, or if a fixed reflectance factor applicable to all wavelengths is stored in the filter.properties attribute of the filter_spct object.

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

See Also

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

Examples

```r
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 and Tfr or A computed for the requested thickness.
**convertTimeUnit**

**Note**

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

**See Also**

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

**Examples**

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

x possibly with the time.unit attribute modified

**Note**

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

See Also
Other time attribute functions: checkTimeUnit(), convertTfrType(), convertThickness(), getTimeUnit(), setTimeUnit()

Examples

my.spct <- sun.spct
my.spct
convolveTimeUnit(my.spct, "day")
my.spct

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

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

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

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

See Also
Other math operators and functions: MathFun, \(^\_\).generic_spct(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct
Description

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

Usage

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

Arguments

x, y R objects
which character Names of attributes to copy, if NULL all those relevant according to the class of x is used as 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)

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

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

**Usage**

cps2irrad(x.sample, pre.fun = NULL, missing.pixs = numeric(0), ...)
cps2Rfr(x.sample, x.white, x.black = NULL, dyn.range = NULL)
cps2Tfr(x.sample, x.clear, x.opaque = NULL, dyn.range = NULL)

**Arguments**

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

**Value**

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

**Note**

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

---

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

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

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

night_length(
  date = lubridate::now(tzone = "UTC"),
  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 matching the default for geocode.
- **tz**: character vector indicating time zone to be used in output and to interpret Date values passed as argument to date.
- **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_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 NOAAs 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" condi-
tions. 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 'fishmethods'
(= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times. The times
returned by the functions in package 'photobiology' have been validated against the NOAA base
implementation.

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
The primary source for the algorithm used is the book: Meeus, J. (1998) Astronomical Algorithms,

A different implementation is available at https://apps-nefsc.fisheries.noaa.gov/AstroCalc4R/
and in R package 'fishmethods'. 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://gml.noaa.gov/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(), 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.
- 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'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

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

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

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

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

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

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

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

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

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

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

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

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

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

### 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)

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

Note

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

See Also

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

Examples

white_led.raw_spct[120:125, ]

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

```r
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
```

```r
```

```r
data(diffraction_single_slit)
```
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**

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

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

**Arguments**

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

**Value**

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

---

**div-.generic_spct**  
**Arithmetic Operators**

**Description**

Integer-division operator for generic spectra.

**Usage**

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

**Arguments**

- `e1`  
  an object of class "generic_spct"
- `e2`  
  an object of class "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.

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
**drop_user_cols**

**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()`, `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**

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

---

**drop_user_cols** *Drop user columns*

**Description**

Remove from spectral object additional columns that are user defined.

**Usage**

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

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

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

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

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

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

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

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

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

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

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

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

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

### Arguments

- **x**
  
  An R object

- **keep.also**
  
  character Additional columns to preserve.

- **...**
  
  needed to allow derivation.

### Value

A copy of x possibly with some columns removed.

### Methods (by class)

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

See Also

Other experimental utility functions: collect2mspc(), thin_wl(), uncollect2spct()

---

e2q

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
- action
- byref
- ...
- .parallel

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
- action
- byref
- 
- ...
- .parallel

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

Methods (by class)

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

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`
e2quantum_multipliers  Calculate energy to quantum multipliers

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

Usage
e2quantum_multipliers(w.length, molar = FALSE)

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

Value
A numeric vector of multipliers

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

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

enable_check_spct  Enable or disable checks

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

Usage
enable_check_spct()

disable_check_spct()

set_check_spct(x)
Arguments

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

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**

- `energy_as_default()`
- `photon_as_default()`
- `quantum_as_default()`
- `Tfr_as_default()`
- `Afr_as_default()`
- `A_as_default()`
- `unset_radiation_unit_default()`
- `unset_filter_qty_default()`
- `unset_user_defaults()`

**Value**

Previous value of the modified option.
energy_irradiance

Calculate (energy) irradiance from spectral irradiance

Description

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

Usage

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

Arguments

w.length numeric vector of wavelength [nm].

s.irrad numeric vector of spectral irradiances in [W m\(^{-2}\) nm\(^{-1}\)] or [mol s\(^{-1}\) sm\(^{-2}\) nm\(^{-1}\)] as indicated by the argument passed to unit.in.

w.band waveband.

unit.in character. Allowed values “energy”, and “photon”, or its alias “quantum”.

check.spectrum logical. Flag indicating whether to sanity check input data, default is TRUE.

use.cached.mult logical. Flag indicating whether multiplier values should be cached between calls.

use.hinges logical. Flag indicating whether to insert “hinges” into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

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

See Also

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

Examples

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

---

**energy_ratio**

Energy:energy ratio

**Description**

Energy irradiance ratio between two wavebands for a radiation spectrum.

**Usage**

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

**Arguments**

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

**Value**

a single numeric value giving the unitless energy ratio.
eq_ratio

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

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

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

---

eq_ratio

Energy:photon ratio

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

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

## Default S3 method:
eq_ratio(spct, w.band = 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_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,
  .paropts = NULL
)

Arguments

spct source_spct.

w.band waveband or list of waveband objects.

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

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

use.cached.mult logical Flag telling whether multiplier values should be cached between calls.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

... other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

name.tag character Used to tag the name of the returned values.

attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx character Name of the column with the names of the members of the collection of spectra.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
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 \([J \text{ mol}^{-1}]\).

Methods (by class)

- eq_ratio(default): Default for generic function
- eq_ratio(source_spct): Method for source_spct objects
- eq_ratio(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

```
  eq_ratio(sun.spct, new_waveband(400,700))
```

---

**ET_ref**

Evapotranspiration

Description

Compute an estimate of reference (= potential) evapotranspiration from meteorological data. Evapotranspiration from vegetation includes transpiration by plants plus evaporation from the soil or other wet surfaces. \(ET_0\) is the reference value assuming no limitation to transpiration due to soil water, similar to potential evapotranspiration (PET). An actual evapotranspiration value \(ET\) can be estimated only if additional information on the plants and soil is available.
Usage

ET_ref(
  temperature,
  water.vp,
  wind.speed,
  net.irradiance,
  nighttime = FALSE,
  atmospheric.pressure = 10.13,
  soil.heat.flux = 0,
  method = "FAO.PM",
  check.range = TRUE
)

ET_ref_day(
  temperature,
  water.vp,
  wind.speed,
  net.radiation,
  atmospheric.pressure = 10.13,
  soil.heat.flux = 0,
  method = "FAO.PM",
  check.range = TRUE
)

Arguments

  temperature  numeric vector of air temperatures (C) at 2 m height.
  water.vp     numeric vector of water vapour pressure in air (Pa).
  wind.speed   numeric Wind speed (m/s) at 2 m height.
  net.irradiance numeric Long wave and short wave balance (W/m2).
  nighttime    logical Used only for methods that distinguish between daytime- and nighttime canopy conductances.
  atmospheric.pressure numeric Atmospheric pressure (Pa).
  soil.heat.flux numeric Soil heat flux (W/m2), positive if soil temperature is increasing.
  method       character The name of an estimation method.
  check.range  logical Flag indicating whether to check or not that arguments for temperature are within range of method. Passed to function calls to water_vp_sat() and water_vp_sat_slope().
  net.radiation numeric Long wave and short wave balance (J/m2/day).

Details

Currently three methods, based on the Penmann-Monteith equation formulated as recommended by FAO56 (Allen et al., 1998) as well as modified in 2005 for tall and short vegetation according to
ASCE-EWRI are implemented in function \texttt{ET\_ref()}. The computations rely on data measured according WHO standards at 2 m above ground level to estimate reference evapotranspiration (\(ET_0\)). The formulations are those for ET expressed in mm/h, but modified to use as input flux rates in W/m² and pressures expressed in Pa.

\textbf{Value}

A numeric vector of reference evapotranspiration estimates expressed in mm/h for \texttt{ET\_ref()} and \texttt{ET\_PM()} and in mm/d for \texttt{ET\_ref\_day()}.

\textbf{References}


\textbf{See Also}

Other Evapotranspiration and energy balance related functions: \texttt{net\_irradiance()}

\textbf{Examples}

\begin{verbatim}
# instantaneous
ET_ref(temperature = 20,
       water.vp = water_RH2vp(relative.humidity = 70,
                               temperature = 20),
       wind.speed = 0,
       net.irradiance = 10)

ET_ref(temperature = c(5, 20, 35),
       water.vp = water_RH2vp(70, c(5, 20, 35)),
       wind.speed = 0,
       net.irradiance = 10)

# Hot and dry air
ET_ref(temperature = 35,
       water.vp = water_RH2vp(10, 35),
       wind.speed = 5,
       net.irradiance = 400)

ET_ref(temperature = 35,
       water.vp = water_RH2vp(10, 35),
       wind.speed = 5,
       net.irradiance = 400,
       method = "FAO.PM")

ET_ref(temperature = 35,
       water.vp = water_RH2vp(10, 35),
       wind.speed = 5,
       net.irradiance = 400,
       method = "FAO.PM")
\end{verbatim}
method = "ASCE.PM.short")

ET_ref(temperature = 35,
    water.vp = water_RH2vp(10, 35),
    wind.speed = 5,
    net.irradiance = 400,
    method = "ASCE.PM.tall")

# Low temperature and high humidity
ET_ref(temperature = 5,
    water.vp = water_RH2vp(95, 5),
    wind.speed = 0.5,
    net.irradiance = -10,
    nighttime = TRUE,
    method = "ASCE.PM.short")

ET_ref_day(temperature = 35,
    water.vp = water_RH2vp(10, 35),
    wind.speed = 5,
    net.radiation = 35e6) # 35 MJ / d / m2

---

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]

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

x[i, j, drop = NULL]

## S3 method for class 'solute_spct'

x[i, j, drop = NULL]

## S3 method for class 'object_spct'

x[i, j, drop = NULL]

## S3 method for class 'chroma_spct'

x[i, j, drop = NULL]

## S3 replacement method for class 'generic_spct'

x[i, j] <- value

## S3 replacement method for class 'generic_spct'

x$name <- value

### Arguments

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

### Details

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

### Value

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

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

See Also

`subset` and `trim_spct`

Examples

```r
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

```r
## 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

Description

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

Usage

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

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

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

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

### 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
**e_fluence**

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)**

- e_fluence(default): Default for generic function
- e_fluence(source_spct): Calculate energy fluence from a source_spct object and the duration of the exposure.
- e_fluence(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.mult,
  use.hinges,
  allow.scaled,
  ...
)
```

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

```r
## S3 method for class 'source_spct'
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  ...
)
```
time.unit = NULL,
scale.factor = 1,
wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
use.cached.mult =getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = NULL,
allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...
)

## S3 method for class 'source_mspct'
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult =getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

Arguments

spct an R object.

w.band a list of waveband objects or a waveband object.

quantity character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".

time.unit character or lubridate::duration object.

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

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

use.cached.mult logical indicating whether multiplier values should be cached between calls.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of
names to assign to returned value.

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 compu-
tation 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, con-
taining one column for each waveband object, an index column with the names of the spectra, and
optionally additional columns with metadata values retrieved from the attributes of the member
spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity
they can be re-expressed as relative fractions or percentages. In the case of vector output, names
attribute is set to the name of the corresponding waveband unless a named list is supplied in which
case the names of the list members are used. The `time.unit` attribute is copied from the spectrum
object to the output. Units are as follows: If units are absolute and `time.unit` is second, \([W \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow [W \text{ m}^{-2}]\) If `time.unit` is day, \([J \text{ d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow [J \text{ m}^{-2}]\); if units are relative, fraction of
one or percent.

Methods (by class)

- `e_irrad(default)`: Default for generic function
- `e_irrad(source_spct)`: Calculates energy irradiance from a source_spct object.
- `e_irrad(source_mspct)`: Calculates energy 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 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()`, `fluence()`, `irrad()`, `q_fluence()`, `q_irrad()`
**e_ratio**

### Examples

```r
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:

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

### Arguments

- **spct**: source_spct
- **w.band.num**: waveband object or a list of waveband objects used to compute the numerator(s) of the ratio(s).
- **w.band.denom**: waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.cached.mult**: logical Flag telling whether multiplier values should be cached between calls.
e_ratio

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)

- e_ratio(default): Default for generic function
- e_ratio(source_spct): Method for source_spct objects
- e_ratio(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.
See Also

Other photon and energy ratio functions: \texttt{eq\_ratio()}, \texttt{q\_ratio()}, \texttt{qe\_ratio()}

Examples

\begin{verbatim}
e_ratio(sun.spct, new_waveband(400,500), new_waveband(400,700))
\end{verbatim}

\begin{verbatim}
e\_response

\textbf{e\_response} \hspace{1cm} \textit{Energy-based photo-response}

\textbf{Description}

This function returns the mean, total, or contribution of response for each waveband and a response spectrum.

\textbf{Usage}

\begin{verbatim}
e\_response(
    spct,  
    w\_band,  
    quantity,  
    time\_unit,  
    scale\_factor,  
    wb\_trim,  
    use\_hinges,
    ...
)
\end{verbatim}

\texttt{## Default S3 method:}
e\_response(
    spct,  
    w\_band,  
    quantity,  
    time\_unit,  
    scale\_factor,  
    wb\_trim,  
    use\_hinges,
    ...
)

\texttt{## S3 method for class 'response\_spct'}
e\_response(
    spct,  
    w\_band = NULL,  
    quantity = "total",  
    time\_unit = NULL,
    ...
)
\end{verbatim}
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",
  ...
)

Arguments

- **spct**: an R object.
- **w.band**: waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- **quantity**: character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
- **time.unit**: character or lubridate::duration object.
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments (possibly used by derived methods).
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx character Name of the column with the names of the members of the collection of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

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

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

Methods (by class)

- e_response(default): Default method for generic function
- e_response(response_mspct): Calculates energy response from a response_mspct

Note

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

See Also

Other response functions: q_response(), response()

Examples

e_response(ccd.spct, new_waveband(200,300))
e_response(photodiode.spct)
FEL.BN.9101.165

Data for typical calibration lamps

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

Format
A numeric vector.

Author(s)
Lasse Ylianttila (data)

FEL_spectrum
Incandescent "FEL" lamp emission spectrum

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

Usage
FEL_spectrum(w.length, k = photobiology::FEL.BN.9101.165, fill = NA_real_)

Arguments
- w.length: numeric vector of wavelengths (nm) for output
- k: a numeric vector with n constants for the function
- fill: if NA, no extrapolation is done, and NA is returned for wavelengths outside the range 250 nm to 900 nm. If NULL then the tails are deleted. If 0 then the tails are set to zero, etc. NA is default.

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

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

**Examples**

```r
FEL_spectrum(400)
FEL_spectrum(250:900)
```

---

**findMultipleWl**  
*Find repeated w.length values*

**Description**

Find repeated w.length values

**Usage**

```r
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) below 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**

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

find_wls(
  x,
  target = NULL,
  col.name.x = NULL,
  col.name = NULL,
  .fun = `<=`,
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE
)

Arguments

x          an R object

(target numeric or character. A numeric value indicates the spectral quantity value for which wavelengths are to be searched. A character representing a number is converted to a number. A character value representing a number followed by a function name, will be also accepted and decoded, such that "0.1max" is interpreted as targeting one tenth of the maximum value in a column. The character strings "half.maximum" and "HM" are synonyms for "0.5max" while "half.range" and "HR" are synonyms for "0.5range". These synonyms are converted to the canonical form before saving them to the returned value.

col.name.x character The name of the column in which to the independent variable is stored. Defaults to "w.length" for objects of class "generic_spct" or derived.

col.name character The name of the column in which to search for the target value.

.fun function A binary comparison function or operator.

.interpolate logical Indicating whether the nearest wavelength value in x should be returned or a value calculated by linear interpolation between wavelength values straddling the target.

.idfactor logical or character Generates an index column of factor type. If idfactor = TRUE then the column is auto named target.idx. Alternatively the column name can be directly passed as argument to idfactor as a character string.

.na.rm logical indicating whether NA values should be stripped before searching for the target.
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.

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(whiteLed.source_spt)
find_wls(whiteLed.source_spt, target = "0.5max")
find_wls(whiteLed.source_spt, target = 0.4)
find_wls(whiteLed.source_spt, target = 0.4, interpolate = TRUE)
find_wls(whiteLed.source_spt, target = c(0.3, 0.4))
find_wls(whiteLed.source_spt, target = c(0.3, 0.4), idfactor = "target")
find_wls(whiteLed.source_spt, target = c(0.3, 0.4), idfactor = TRUE)
find_wls(whiteLed.source_spt, target = "0.5max")
find_wls(whiteLed.source_spt, target = "0.85max")
find_wls(whiteLed.source_spt, target = "0.5range")

led.df <- as.data.frame(whiteLed.source_spt)
find_wls(led.df)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length")
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = 0.4)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = c(0.3, 0.4))
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = 0.4, idfactor = "target")
```

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,
)```
### fit_peaks

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

```r
fit_valleys(
  x,
  valleys.idx,
  span,
  x.col.name = NULL,
y.col.name,
method,
max.span = 5L,
maximum = FALSE,
keep.cols = NULL
)
```

#### Arguments

- **x**
  - generic_spct or data.frame object.
- **peaks.idx, valleys.idx**
  - logical or integer Indexes into x selecting global or local extremes.
- **span**
  - odd integer The span used when refining the location of maxima or minima of x.
- **x.col.name, y.col.name**
  - character Name of the column of x on which to operate.
- **method**
  - character The method to use for the fit.
- **max.span**
  - odd integer The maximum number of data points used when when refining the location of maxima and minima.
- **maximum**
  - logical A flag indicating whether to search for maxima or minima.
- **keep.cols**
  - logical Keep unrecognized columns in data frames

#### 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

```r
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

**Fluence**

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

#### Usage

```r
fluence(
  spct,      # spectrum
  w.band,    # waveband range
  unit.out,  # output unit
  exposure.time,  # exposure time
  scale.factor,  # scaling factor
  wb.trim,    # waveband trimming
  use.cached.mult,  # use cached multiplication factors
  use.hinges,  # use hinges for fitting
  allow.scaled,  # allow scaled output
  ...
)
```

## Default S3 method:

```r
fluence(
  spct,      # spectrum
  w.band,    # waveband range
  unit.out,  # output unit
  exposure.time,  # exposure time
  scale.factor,  # scaling factor
  wb.trim,    # waveband trimming
  use.cached.mult,  # use cached multiplication factors
  use.hinges,  # use hinges for fitting
  allow.scaled,  # allow scaled output
  ...
)
```

## S3 method for class 'source_spct'

```r
fluence(      # spectrum
  spct,
  ...)
Arguments

**spct**

an R object.

**w.band**

ea list of waveband objects or a waveband object.

**unit.out**

character string with allowed values "energy", and "photon", or its alias "quantum".

**exposure.time**

lubridate::duration object.

**scale.factor**

numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.

**wb.trim**

logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.

**use.cached.mult**

logical indicating whether multiplier values should be cached between calls.

**use.hinges**

logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.

... other arguments (possibly used by derived methods).
naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx character Name of the column with the names of the members of the collection of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, \([\text{W m}^{-2} \text{ nm}^{-1}] \rightarrow \text{[mol s}^{-1} \text{ m}^{-2}\] If time.unit is day, \([\text{J d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow \text{[mol d}^{-1} \text{ m}^{-2}]\]

Methods (by class)

- fluence(default): Default for generic function
- fluence(source_spct): Calculate photon fluence from a source_spct object and the duration of the exposure
- fluence(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") )
format.solar_time

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(), sun_angles()

format.tod_time

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 Time of day functions: as_tod(), print.tod_time()
formatted_range  
*Compute range and format it*

**Description**

Compute the range of an R object, and format it as string suitable for printing.

**Usage**

```r
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**

```r
formatted_range(c(1, 3.5, -0.01))
```

---

fscale  
*Rescale a spectrum using a summary function*

**Description**

These methods return a spectral object of the same class as the one supplied as argument but with the spectral data rescaled based on a summary function `f` applied over a specific range of wavelengths and a target value for the summary value.

**Usage**

```r
fscale(x, ...)
```

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

```
## S3 method for class 'source_spct'
fscale(
```

```r
```
fscale

x,
range = NULL,
f = "mean",
target = 1,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
set.scaled = target == 1,
...
)

## S3 method for class 'response_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...
)

## S3 method for class 'filter_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  set.scaled = target == 1,
  ...
)

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

## S3 method for class 'solute_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
fscale

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

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

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

### Arguments

- **x** An R object
- **...** additional named arguments passed down to `f`
- **range** numeric. An R object on which `range()` returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm)
- **f** character string. "mean" or "total" for scaling so that this summary value becomes 1 for the returned object, or the name of a function taking `x` as first argument and returning a numeric value.
- **target** numeric A constant used as target value for scaling.
- **unit.out** character. Allowed values "energy", and "photon", or its alias "quantum".
- **set.scaled** logical or NULL Flag indicating if the data is to be marked as "scaled" or not.
- **qty.out** character. Allowed values "transmittance", and "absorbance".
- **col.names** character vector containing the names of columns or variables to which to apply the scaling.
- **.parallel** logical if TRUE, apply function in parallel, using parallel backend provided by foreach.
a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

After scaling, applying the function passed as argument to \( f \) to the scaled spectrum will return the value passed as argument to target. The default for set.scaled depends dynamically on the passed to target. Sometimes we rescale a spectrum to a "theoretical" value for the summary, while in other cases we rescale the spectrum to a real-world target value of e.g. a reference energy irradiance. In the first case we say that the data are expressed in relative units, while in the second case we retain actual physical units. To indicate this, this package uses an attribute, which will by default be set assuming the first of these two situations when target == 1 and not set assuming the second situation otherwise. These defaults can be overridden with an explicit logical argument passed to set.scaled.

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)

- `fscale` (default): Default for generic function
- `fscale` (source_spct):
- `fscale` (response_spct):
- `fscale` (filter_spct):
- `fscale` (reflector_spct):
- `fscale` (solute_spct):
- `fscale` (raw_spct):
- `fscale` (cps_spct):
- `fscale` (generic_spct):
- `fscale` (source_mspct):
- `fscale` (response_mspct):
- `fscale` (filter_mspct):
- `fscale` (reflector_mspct):
- `fscale` (solute_mspct):
- `fscale` (raw_mspct):
- `fscale` (cps_mspct):
- `fscale` (generic_mspct):
Important changes

Metadata describing the rescaling operation are stored in an attribute only if `set.scaled = TRUE` is passed to the call. The exact format and data stored in the attribute "scaled" has changed during the development of the package. Spectra re-scaled with earlier versions will lack some information. To obtain the metadata in a consistent format irrespective of this variation use accessor `getScaling()`, which fills missing fields with `NA`.

Note

Method `fscale` is not implemented for `solute_spct` objects as the spectral data stored in them are a description of an intensive property of a substance. To represent solutions of specific concentrations of solutes, `filter_spct` objects can be used.

See Also

Other rescaling functions: `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 <- fscale(sun.spct,
       f = e_irrad,
       range = c(400, 700),
       target = 400)  # a target in W m^-2
s400.spct
e_irrad(s400.spct, c(400, 700))
```

---

**fshift**  
*Shift the scale of a spectrum using a summary function*

Description

The `fshift()` methods return a spectral object of the same class as the one supplied as argument but with the spectral data on a zero-shifted scale. A range of wavelengths is taken as a zero reference and the summary calculated with `f` for this waveband is subtracted. This results in a zero shift (= additive correction) to the values in the returned object. Metadata attributes are retained unchanged.
Usage

fshift(x, ...)

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

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

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

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

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

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

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

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

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

## S3 method for class 'response_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
fshift(
  x,
  range = c(wl_min(x), wl_min(x) + 10),
  f = "min",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)
  .parallel = FALSE,
  .paropts = NULL
)

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

A copy of \(x\) with the spectral data values replaced with values zero-shifted.

a new object of the same class as \(x\).

Methods (by class)

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

Note

Method \(fshift\) is not implemented for `solute_spct` objects as the spectral data stored in them are a description of an intensive property of a substance. To represent solutions of specific concentrations of solutes, `filter_spct` objects can be used.

See Also

Other rescaling functions: `fscale()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setNormalized()`, `setScaled()`
**generic_mspct**

*Collection-of-spectra constructor*

**Description**

Converts a list of spectral objects into a "multi spectrum" object by setting the class attribute of the list of spectra to the corresponding multi-spct class, check that components of the list belong to the expected class.

**Usage**

```
generic_mspct(
  l = NULL,
  class = "generic_spct",
  ncol = 1,
  byrow = FALSE,
  dim = c(length(l)/%/%ncol, ncol)
)
```

calibration_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

raw_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

cps_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

source_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

filter_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

reflector_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

object_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

solute_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

response_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

chroma_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

**Arguments**

- **l** list of generic_spct or derived classes
- **class** character The multi spectrum object class or the expected class for the elements of l
- **ncol** integer Number of ‘virtual’ columns in data
- **byrow** logical If ncol > 1 how to read in the data
getBSWFUsed

Get the "bswf.used" attribute

Description

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

Usage

getBSWFUsed(x)

Arguments

x a source_spct object

Value

character string
getFilterProperties

Note

if x is not a source_spct object, NA is returned

See Also

Other BSWF attribute functions: setBSWFUsed()

Examples

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.

Usage

getFilterProperties(x, return.null, ...)  filter_properties(x, return.null, ...)

## Default S3 method:
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'summary_filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'generic_mspct'
getFilterProperties(x, return.null = FALSE, ..., idx = "spct.idx")

Arguments

x a filter_spct object

return.null logical If true, NULL is returned if the attribute is not set, otherwise the expected list is returned with all fields set to NA.

... Allows use of additional arguments in methods for other classes.

idx character Name of the column with the names of the members of the collection of spectra.
getHowMeasured

Value

A list with fields named "Rfr.constant" [1], "thickness" [m] and "attenuation.mode". If the attribute is not set, and return.null is FALSE, a list with fields set to NA is returned, otherwise, NULL.

Methods (by class)

- getFilterProperties(default): default
- getFilterProperties(filter_spct): generic_spct
- getFilterProperties(summary_filter_spct): summary_generic_spct
- getFilterProperties(generic_mspct): filter_mspct

Note

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

See Also


Examples

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

getHowMeasured(x, ...)

how_measured(x, ...)

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

## 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)

- getHowMeasured(default): default
- getHowMeasured(generic_spct): `generic_spct`
- getHowMeasured(summary_generic_spct): `summary_generic_spct`
- getHowMeasured(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)`
getIdFactor

Get the "idfactor" attribute

Description
Function to read the "idfactor" attribute of an existing generic_spct.

Usage
getIdFactor(x)

Arguments
x  a generic_spct object

Value
character

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

See Also
Other idfactor attribute functions: setIdFactor()

Examples
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
getInstrSettings

Value

list (depends on instrument type)

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrSettings(), getSoluteProperties(), getWhatMeasured(), getWhereMeasured(),
get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(),
setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(),
trimInstrDesc(), trimInstrSettings()
Description
Function to read the "K.type" attribute of an existing solute_spct object.

Usage
getKType(x)

Arguments
x
a solute_spct object

Value
character string

Note
If x is not a solute_spct or a summary_solute_spct object, NA is returned.

See Also
Other K attribute functions: setKType()

Examples
print("missing example")

Description
Function to read the "mspct.version" attribute of an existing generic_mspct object.

Usage
getMspctVersion(x)

Arguments
x
a generic_mspct object
getMultipleWl

Value

numeric value

Note

if x is not a generic_mspct object, NA is returned, and if it the attribute is missing, zero is returned with a warning.

getMultipleWl (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)
Description

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

Usage

getNormalized(x, .force.numeric = FALSE)
getNormalised(x, .force.numeric = FALSE)
getNormalization(x)
getNormalisation(x)

Arguments

x
a generic_spct object

.force.numeric
logical If TRUE always silently return a numeric value, with FALSE encoded as zero, and character values as NA.

Value

numeric or logical (possibly character for objects created with earlier versions).

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()

Examples

sun_norm.spct <- normalize(sun.spct)
getNormalized(sun.spct)
getNormalization(sun.spct)
**getResponseType**  
*Get the "response.type" attribute*

**Description**
Function to read the "response.type" attribute of an existing response_spct object.

**Usage**

```r
getResponseType(x)
```

**Arguments**

- `x`  
a response_spct object

**Details**
Objects of class response_spct() can contain data for a response spectrum or an action spectrum. Response spectra are measured using the same photon (or energy) irradiance at each wavelength. Action spectra are derived from dose response curves at each wavelength, and responsivity at each wavelength is expressed as the reciprocal of the photon fluence required to obtain a fixed level of response.

**Value**

character string

**Note**

If `x` is not a response_spct object, NA is returned.

**Examples**

```r
getResponseType(ccd.spct)
getSocketAddress(sun.spct)
```
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

Value

character string

Note

if x is not a filter_spct object, NA is returned

See Also

Other Rfr attribute functions: setRfrType()

Description

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

Usage

getScaled(x, .force.list = FALSE)

getScaling(x)

Arguments

x a generic_spct object

.force.list logical If TRUE always silently return a list, with FALSE encoded field multiplier = 1.
getSoluteProperties

Value
logical

Note
if x is not a filter_spct object, NA is returned

See Also
Other rescaling functions: fscale(), fshift(), getNormalized(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

Examples

scaled.spct <- fscale(sun.spct)
getScaled(scaled.spct)

getSoluteProperties Get the "solute.properties" attribute

Description
Function to read the "solute.properties" attribute of an existing solute_spct or a solute_mspct objects.

Usage
getSoluteProperties(x, return.null, ...)
solute_properties(x, return.null, ...)

## Default S3 method:
getSoluteProperties(x, return.null = FALSE, ...)

## S3 method for class 'solute_spct'
getSoluteProperties(x, return.null = FALSE, ...)

## S3 method for class 'summary_solute_spct'
getSoluteProperties(x, return.null = FALSE, ...)

## S3 method for class 'solute_mspct'
getSoluteProperties(x, return.null = FALSE, ..., idx = "spct.idx")
getSoluteProperties

Arguments

- **x**
  - solute_spct A spectrum of coefficients of attenuation.
- **return.null**
  - logical If true, NULL is returned if the attribute is not set, otherwise the expected list is returned with all fields set to NA.
- **idx**
  - character Name of the column with the names of the members of the collection of spectra.

Value

A list with fields named "mass", "formula", "structure", "name" and "ID". If the attribute is not set, and `return.null` is FALSE, a list with fields set to NA is returned, otherwise, NULL.

Methods (by class)

- getSoluteProperties(default): default
- getSoluteProperties(solute_spct): solute_spct
- getSoluteProperties(summary_solute_spct): summary_solute_spct
- getSoluteProperties(solute_mspct): solute_mspct

Note

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

See Also


Examples

```r
solute_properties(water.spct)
```
getSpctVersion

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

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.
getTimeUnit

See Also

Other Tfr attribute functions: setTfrType()

Examples

getTimeUnit(polyester.spct)

g getTimeUnit

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

Description

Function to read the "time.unit" attribute

Usage

gTimeUnit(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

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

...  
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)

- getWhatMeasured(default): default
- getWhatMeasured(generic_spct): generic_spct
- getWhatMeasured(summary_generic_spct): summary_generic_spct
- getWhatMeasured(generic_mspct): generic_mspct

Note

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

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhenMeasured(), getWhenMeasured(),
get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(),
setWhatMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(),
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)

- getWhenMeasured(default): default
- getWhenMeasured(generic_spct): generic_spct
- getWhenMeasured(summary_generic_spct): summary_generic_spct
- getWhenMeasured(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)
## 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", .bind.geocodes = TRUE)

### 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.
- **.bind.geocodes**: logical In the case of collections of spectra if .bind.geocodes = TRUE, the default, the returned value is a single geocode with one row for each member spectrum. Otherwise the individual geocode data frames are returned in a list column within a tibble.

### Value

A data.frame with a single row and at least columns "lon" and "lat", unless expand is set to FALSE.

### Methods (by class)

- getWhereMeasured(default): default
- getWhereMeasured(generic_spct): generic_spct
- getWhereMeasured(summary_generic_spct): summary_generic_spct
- getWhereMeasured(generic_mspct): generic_mspct

### Note

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

### See Also

get_attributes

Get the metadata attributes

Description
Method returning attributes of an object of class generic_spct or derived, or of class waveband. Only attributes defined and/or set by package 'photobiology' for objects of the corresponding class are returned. Parameter which can be used to subset the list of attributes.

Usage

get_attributes(x, which, ...)

## S3 method for class 'generic_spct'
get_attributes(x, which = NULL, allowed = all.attributes, ...)

## S3 method for class 'source_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'filter_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'reflector_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'object_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'solute_spct'
get_attributes(x, which = NULL, ...)

## S3 method for class 'waveband'
get_attributes(x, which = NULL, ...)

Arguments

x a generic_spct object.

which character vector Names of attributes to retrieve.

... currently ignored

allowed character vector Names of attributes accepted by which.

Details
Vectors of character strings passed as argument to which are parsed so that if the first member string is "-" the remaining members are removed from the allowed; and if it is "=" the remaining members are used if in allowed. If the first member is none of these three strings, the behaviour
is the same as if the first string is "=". If which is NULL all the attributes in allowed are used. The string "" means no attributes, and has precedence over any other values in the character vector. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

Named list of attribute values.

Methods (by class)

- get_attributes(generic_spct): generic_spct
- get_attributes(source_spct): source_spct
- get_attributes(filter_spct): filter_spct
- get_attributes(reflector_spct): reflector_spct
- get_attributes(object_spct): object_spct
- get_attributes(solute_spct): solute_spct
- get_attributes(waveband): waveband

See Also

select_spct_attributes


get_peaks: 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 splus2R.

Usage

get_peaks(
  x,
  y,
  ignore_threshold = 0,
  span = 5,
get_peaks

strict = TRUE,
x_unit = "",
x_digits = 3,
na.rm = FALSE
)

get_valleys(
  x,
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()
green_leaf.spct

Examples

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

---

**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**


**Examples**

```r
green_leaf.spct```

---


**head_tail**  

*Return the First and Last Parts of an Object*

---

**Description**

Returns the first and last "parts" (rows or members) of a spectrum, dataframe, vector, function, table or `ftable`. In other words, the combined output from methods `head` and `tail`.

**Usage**

```r
head_tail(x, n, ...)
```

## Default S3 method:

```r
head_tail(x, n = 3L, ...)
```

## S3 method for class 'data.frame'

```r
head_tail(x, n = 3L, ...)
```

## S3 method for class 'matrix'

```r
head_tail(x, n = 3L, ...)
```

## S3 method for class 'function'

```r
head_tail(x, n = 6L, ...)
```

## S3 method for class 'table'

```r
head_tail(x, n = 6L, ...)
```

## S3 method for class 'ftable'

```r
head_tail(x, n = 6L, ...)
```

**Arguments**

- `x`  
  an R object.

- `n`  
  integer. If positive, `n` rows or members in the returned object are copied from each of "head" and "tail" of `x`. If negative, all except `n` elements of `x` from each of "head" and "tail" are returned.

- `...`  
  arguments to be passed to or from other methods.

**Details**

The value returned by `head_tail()` is equivalent to row binding the the values returned by `head()` and `tail()`, although not implemented in this way. The same specializations as defined in package 'utils' for `head()` and `tail()` have been implemented.

**Value**

An object (usually) like `x` but smaller, except when `n = 0`. For `ftable` objects `x`, a transformed `format(x)`.
Methods (by class)

- head_tail(default):
- head_tail(data.frame):
- head_tail(matrix):
- head_tail('function'):
- head_tail(table):
- head_tail(ftable):

Note

For some types of input, like functions, the output may be confusing, however, we have opted for consistency with existing functions. The code is in part a revision of that of head() and tail() from package ‘utils’. This method is especially useful when checking spectral data, as both ends are of interest.

head_tail() methods for function, table and ftable classes, are wrappers for head() method.

See Also

head, and compare the examples and the values returned to the examples below.

Examples

```r
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

```r
insert_hinges(x, y, h)
```
insert_spct_hinges

Arguments

  x  numeric vector (sorted in increasing order)
  y  numeric vector
  h  a numeric vector giving the wavelengths at which the y values should be inserted
      by interpolation, no interpolation is indicated by an empty vector (numeric(0))

Value

  a data.frame with variables x and y. Unless the hinge values were already present in y, each
  inserted hinge, expands the vectors returned in the data frame by one value.

Note

  Insertion is a costly operation but I have tried to optimize this function as much as possible by
  avoiding loops. Earlier this function was implemented in C++, but a bug was discovered and I have
  now rewritten it using R.

See Also

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

Examples

  with(sun.data,
      insert_hinges(w.length, s.e.irrad,
                    c(399.99, 400.00, 699.99, 700.00)))
integrate_spct

Arguments

spct an object of class "generic_spct"
hinges numeric vector of wavelengths (nm) at which the s.irrad should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0))
byref logical indicating if new object will be created by reference or by copy of spct

Value

a generic_spct or a derived type with variables w.length and other numeric variables.

Note

Inserting wavelengths values "hinges" immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data has a large wavelength step size.

Examples

insert_spct_hinges(sun.spct, c(399.99,400.00,699.99,700.00))
insert_spct_hinges(sun.spct,
c(199.99,200.00,399.50,399.99,400.00,699.99,
    700.00,799.99,1000.00))

integrate_spct Integrate spectral data.

Description

This function gives the result of integrating spectral data over wavelengths.

Usage

integrate_spct(spct)

Arguments

spct generic_spct

Value

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

**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**

```r
integrate_xy(x, y)
```

**Arguments**

- `x`: numeric vector.
- `y`: numeric vector.

**Value**

A single numeric value with no change in scale factor: e.g. \([\text{W m}^{-2} \text{ nm}^{-1}] \rightarrow [\text{W m}^{-2}]\)

**See Also**

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

**Examples**

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

Map a spectrum to new wavelength values.

Description

This function gives the result of interpolating spectral data from the original set of wavelengths to a new one.

Usage

interpolate_spct(spct, w.length.out = NULL, fill = NA, length.out = NULL)

interpolate_mspct(
  mspct,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

Arguments

spct  
generic_spct

w.length.out  
numeric vector of wavelengths (nm)

fill  
a value to be assigned to out of range wavelengths

length.out  
numeric value

mspct  
an object of class "generic_mspct"

.parallel  
if TRUE, apply function in parallel, using parallel backend provided by foreach

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

Details

If length.out 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.
interpolate_spectrum

**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_spct(sun.spct, 400:500, NA)
interpolate_spct(sun.spct, 400:500, NULL)
interpolate_spct(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_spct(sun.spct, c(400,500), length.out=201)
```

**interpolate_spectrum  Calculate spectral values at a different set of wavelengths**

**Description**

Interpolate/re-express spectral irradiance (or other spectral quantity) values at new wavelengths values. This is a low-level function operating on numeric vectors and called by higher level functions in the package, such as mathematical operators for classes for spectral data.

**Usage**

```r
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

```r
my.w.length <- 300:700
with(sun.data, interpolate_spectrum(w.length, s.e.irrad, my.w.length))
```

```r
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

```r
interpolate_wl(x, w.length.out, fill, length.out, ...)
```

```
## Default S3 method:
interpolate_wl(x, w.length.out, fill, length.out, ...)
```

```
## S3 method for class 'generic_spct'
interpolate_wl(x, w.length.out = NULL, fill = NA, length.out = NULL, ...)
```

```
## S3 method for class 'generic_mspct'
interpolate_wl(
  x,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  ..., .parallel = FALSE,
  .paropts = NULL
)
```
interpolate_wl

Arguments

- **x**: an R object
- **w.length.out**: numeric vector of wavelengths (nm)
- **fill**: a value to be assigned to out of range wavelengths
- **length.out**: numeric value
- **...**: not used
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

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)

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

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

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),
  ...
Arguments

 spat
 an R object.
 w.band
 waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are summarized.
 unit.out
 character Allowed values "energy", and "photon", or its alias "quantum".
 quantity
 character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
 time.unit
 character or lubridate::duration object.
 scale.factor
 numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
 wb.trim
 logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
 use.cached.mult
 logical indicating whether multiplier values should be cached between calls.
 use.hinges
 logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
 allow.scaled
 logical indicating whether scaled or normalized spectra as argument to spat 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)
• irrad(default): Default for generic function
• irrad(source_spct): Calculates irradiance from a source_spct object.
• irrad(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 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.
irradiance

See Also
Other irradiance functions: e_fluence(), e_irrad(), fluence(), q_fluence(), q_irrad()

Examples
irrad(sun.spct, waveband(c(400,700)))
irrad(sun.spct, waveband(c(400,700)), "energy")
irrad(sun.spct, waveband(c(400,700)), "photon")
irrad(sun.spct, split_bands(c(400,700), length.out = 3))
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")

irradiance

Photon or energy irradiance from spectral energy or photon irradiance.

Description
Energy or photon irradiance for one or more wavebands of a radiation spectrum.

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

Arguments
w.length numeric Vector of wavelength [nm].
s.irrad numeric vector of spectral (energy) irradiances [W m\(^{-2}\) nm\(^{-1}\)].
w.band waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are summarized.
unit.out, unit.in character Allowed values "energy", and "photon", or its alias "quantum".
check.spectrum logical Flag indicating whether to sanity check input data, default is TRUE.
**irrad_extraterrestrial**

**Description**

Estimate of down-welling solar (short wave) irradiance at the top of the atmosphere above a location on Earth, computed based on angles, Sun-Earth distance and the solar constant. Astronomical computations are done with function `sun_angles()`.

**Examples**

```r
with(sun.data, irradiance(w.length, s.e.irrad, new_waveband(400,700), "photon"))
```

---

**irrad_extraterrestrial**

*Extraterrestrial 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 single numeric value or a vector of numeric values with no change in scale factor: \([mol \, s^{-1} \, sm^{-2} \, nm^{-1}]\)

yields \([mol \, s^{-1} \, sm^{-2}]\)

**Note**

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum()` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector. The is no reason for setting `use.cpp.code=FALSE` other than for testing the improvement in speed, or in cases where there is no suitable C++ compiler for building the package.

**See Also**

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

Usage

```r
irrad_extraterrestrial(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  solar.constant = "NASA"
)
```

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.

solar.constant numeric or character If character, "WMO" or "NASA", if numeric, an irradiance value in the same units as the value to be returned.

Value

Numeric vector of extraterrestrial irradiance (in W / m2 if solar constant is a character value).

See Also

Function `sun_angles`.

Examples

```r
library(lubridate)

irrad_extraterrestrial(ymd_hm("2021-06-21 12:00", tz = "UTC"))
irrad_extraterrestrial(ymd_hm("2021-12-21 20:00", tz = "UTC"))
irrad_extraterrestrial(ymd_hm("2021-06-21 00:00", tz = "UTC") + hours(1:23))
```

---

is.generic_mspct  Query class of spectrum objects

Description

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

is.generic_mspct(x)

is.calibration_mspct(x)

is.raw_mspct(x)

is.cps_mspct(x)

is.source_mspct(x)

is.response_mspct(x)

is.filter_mspct(x)

is.reflector_mspct(x)

is.object_mspct(x)

is.solute_mspct(x)

is.chroma_mspct(x)

is.any_mspct(x)

Arguments

x an R object.

Value

These functions return TRUE if its argument is a of the queried type of spectrum and FALSE otherwise.

Note

Derived types also return TRUE for a query for a base type such as generic_mspct.

Examples

my.mspct <- filter_mspct(list(polyester.spct, yellow_gel.spct))
is.any_mspct(my.mspct)
is.filter_mspct(my.mspct)
is.source_mspct(my.mspct)
is.generic_spct

Query class of spectrum objects

Description

Functions to query whether an object is of a given type of spectrum.

Usage

is.generic_spct(x)
is.raw_spct(x)
is.calibration_spct(x)
is.cps_spct(x)
is.source_spct(x)
is.response_spct(x)
is.filter_spct(x)
is.reflector_spct(x)
is.object_spct(x)
is.solute_spct(x)
is.chroma_spct(x)
is.any_spct(x)

Arguments

x an R object.

Value

A logical value, TRUE if the argument passed to x is an object of the queried type of spectrum and FALSE otherwise.

Note

Derived types also return TRUE for a query for a base type such as generic_spct, following R’s practice.
Examples

```r
is.source_spct(sun.spct)
is.filter_spct(sun.spct)
is.generic_spct(sun.spct)
```

```r
is.source_spct(sun.spct)
is.filter_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

```r
is.old_spct(object)
```

#### Arguments

- `object` an R object

#### Value

logical

#### See Also

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

---

### is.solar_time

*Query class*

#### Description

Query class

#### Usage

```r
is.solar_time(x)
is.solar_date(x)
```
is.summary_generic_spct

Arguments
   x        an R object.

See Also
   Other Local solar time functions: as.solar_date(), print.solar_time(), solar_time()

is.summary_generic_spct
   Query class of spectrum summary objects

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

Usage
   is.summary_generic_spct(x)
   is.summary_raw_spct(x)
   is.summary_cps_spct(x)
   is.summary_source_spct(x)
   is.summary_response_spct(x)
   is.summary_filter_spct(x)
   is.summary_reflector_spct(x)
   is.summary_object_spct(x)
   is.summary_solute_spct(x)
   is.summary_chroma_spct(x)
   is.any_summary_spct(x)

Arguments
   x        an R object.

Value
   These functions return TRUE if its argument is a of the queried type of spectrum and FALSE otherwise.
Note

Derived types also return TRUE for a query for a base type such as generic_spct.

Examples

```r
sm <- summary(sun.spct)
is.summary_source_spct(sm)
```

<table>
<thead>
<tr>
<th>is.waveband</th>
<th>Query if it is a waveband</th>
</tr>
</thead>
</table>

Description

Functions to check if an object is waveband.

Usage

```r
is.waveband(x)
```

Arguments

- `x`: any R object

Value

`is.waveband` returns TRUE if its argument is a waveband and FALSE otherwise.

<table>
<thead>
<tr>
<th>isValidInstrDesc</th>
<th>Check the &quot;instr.desc&quot; attribute</th>
</tr>
</thead>
</table>

Description

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

Usage

```r
isValidInstrDesc(x)
```

Arguments

- `x`: a generic_spct object

Value

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

See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(),
getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(),
setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(),
trimInstrDesc(), trimInstrSettings()

isValidInstrSettings  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

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(),
getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setFilterProperties(), setHowMeasured(), setInstrDesc(), setInstrSettings(), setSoluteProperties(),
setWhatMeasured(), setWhenMeasured(), setWhereMeasured(), spct_attr2tb(), spct_metadata(),
trimInstrDesc(), trimInstrSettings()
is_absorbance_based

Query if a spectrum contains absorbance or transmittance data

Description

Functions to query if an filter spectrum contains spectral absorbance data or spectral transmittance data.

Usage

is_absorbance_based(x)

is_absorptance_based(x)

is_transmittance_based(x)

Arguments

x an R object

Value

is_absorbance_based returns a logical value, TRUE if its argument is a filter_spct object that contains spectral absorbance data and FALSE otherwise, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

is_absorptance_based returns a logical value, if its argument is a filter_spct object, TRUE if it contains data as spectral absorptance and FALSE otherwise, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

is_transmittance_based returns TRUE if its argument is a filter_spct object that contains spectral transmittance data and FALSE if it does not contain such data, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

See Also

Other query units functions: is_mole_based(), is_photon_based()

Examples

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

is_absorptance_based(polyester.spct)

is_transmittance_based(polyester.spct)
**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**

```r
is_effective(x)
```

## Default S3 method:

```r
is_effective(x)
```

## S3 method for class 'waveband'

```r
is_effective(x)
```

## S3 method for class 'generic_spct'

```r
is_effective(x)
```

## S3 method for class 'source_spct'

```r
is_effective(x)
```

## S3 method for class 'summary_generic_spct'

```r
is_effective(x)
```

## S3 method for class 'summary_source_spct'

```r
is_effective(x)
```

**Arguments**

- `x` an R object

**Value**

A logical.

**Methods (by class)**

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

See Also
Other waveband attributes: labels(), normalization()

Examples
is_effective(summary(sun.spct))

---

is_mole_based  
Query if a spectrum contains mole or mass based data

Description
Functions to check if an solute attenuation spectrum contains coefficients on expressed on mole of mass base.

Usage
is_mole_based(x)

is_mass_based(x)

Arguments
x  
an R object

Value
is_mole_based returns TRUE if its argument is a solute_spct object that contains spectral K.mole data and FALSE if it contains K.mass data, but returns NA for any other R object, including those belonging other generic_spct-derived classes. is_mass_based returns the complement of is_mole_based.

See Also
Other query units functions: is_absorbance_based(), is_photon_based()

Examples
print("missing example")
**is_normalized**

Query whether a generic spectrum has been normalized.

**Description**

This function tests a generic_spct object for an attribute that signals whether the spectral data has been normalized or not after the object was created.

**Usage**

```r
is_normalized(x)
```

```r
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 query if source_spct and response_spct objects contain photon-based or energy-based data.

**Usage**

```r
is_photon_based(x)
```

```r
is_energy_based(x)
```
Arguments

x         any R object

Value

is_photon_based returns a logical value, TRUE if its argument is a source_spct or a response_spct object that contains photon base data and FALSE otherwise, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

is_energy_based returns a logical value, TRUE if its argument is a source_spct or a response_spct object that contains energy base data and FALSE otherwise, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

See Also

Other query units functions: is_absorbance_based(), is_mole_based()

Examples

colnames(sun.spct)
is_photon_based(sun.spct)
my.spct <- sun.spct[, c("w.length", "s.e.irrad")]
is_source_spct(my.spct)
is_photon_based(my.spct)

colnames(sun.spct)
is_energy_based(sun.spct)
my.spct <- sun.spct[, c("w.length", "s.q.irrad")]
is_source_spct(my.spct)
is_energy_based(my.spct)
is_tagged

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

```r
scaled.spct <- fscale(sun.spct)
is_scaled(sun.spct)
is_scaled(scaled.spct)
```

---

is_tagged  Query if a spectrum is tagged

Description

Functions to check if an spct object contains tags.

Usage

```r
is_tagged(x)
```

Arguments

\( x \)  any R object

Value

`is_tagged` returns a logical value, \( \text{TRUE} \) if its argument is a a spectrum that contains tags and \( \text{FALSE} \) if it is an untagged spectrum, but returns \( \text{NA} \) for any other R object.

See Also

Other tagging and related functions: `tag()`, `untag()`, `wb2rect_spct()`, `wb2spct()`, `wb2tagged_spct()`

Examples

```r
is_tagged(sun.spct)
```
join_mspct

Join all spectra in a collection

Description

Join all the spectra contained in a homogeneous collection, returning a data frame with spectral-data columns named according to the names of the spectra in the collection. By default a full join is done within the overlapping range of wavelengths, after interpolating the spectra to a shared set of wavelength values, and discarding data for wavelength not shared. Alternatively, filling the spectral data for wavelengths outside the overlapping range with with NA when data is not available.

Usage

join_mspct(x, type, ...)

## Default S3 method:
join_mspct(x, type = "full", ...)

## S3 method for class 'generic_mspct'
join_mspct(x, type = "full", col.name, validate.names = TRUE, ...)

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

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

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

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

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

## S3 method for class 'solute_mspct'
join_mspct(x, type = "full", validate.names = TRUE, ...)

Arguments

x  
generic_mspct object, or an object of a class derived from generic_mspct.
type character Type of join: "inner" (default) or "full". See details for more information.
... ignored (possibly used by derived methods).
col.name character, name of the column in the spectra to be preserved, in addition to "w.length".
validate.names logical A flag to enable (default) or disable validation of column names with make.names.
unit.out character Allowed values "energy", and "photon", or its alias "quantum".
qty.out character Allowed values "transmittance", "absorptance", and "absorbance" and in the method for object_spct, also "reflectance".

Value
A data.frame with the spectra joined by, possibly interpolated, wavelength, with rows sorted by wavelength (variable w.length) and data columns named according to the names of members in x, by default made unique and valid.

Methods (by class)
• join_mspct(default):
• join_mspct(generic_mspct):
• join_mspct(source_mspct):
• join_mspct(response_mspct):
• join_mspct(filter_mspct):
• join_mspct(reflector_mspct):
• join_mspct(object_mspct):
• join_mspct(solute_mspct):

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

Examples
my.mspct <- solute_mspct(list(water = water.spct, pha = phenylalanine.spct))
join_mspct(my.mspct, type = "inner")
join_mspct(my.mspct, type = "full")
**Description**

A method specialization that extracts the name and label of objects of class `waveband`.

**Usage**

```r
## S3 method for class 'waveband'
labels(object, ...)
```

```r
## S3 method for class 'generic_spct'
labels(object, ...)
```

**Arguments**

- `object` an object of class "waveband"
- `...` not used in current version

**Methods (by class)**

- `labels(generic_spct):`

**See Also**

Other waveband attributes: `is_effective()`, `normalization()`

**Examples**

```r
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**

- \texttt{w.length} (nm)
- \texttt{Rfr} (0..1)
- \texttt{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: \texttt{A.illuminant.spct}, \texttt{D65.illuminant.spct}, \texttt{Ler_leaf_rflt.spct}, \texttt{Ler_leaf_trns.spct}, \texttt{Ler_leaf_trns_i.spct}, \texttt{black_body.spct}, \texttt{ccd.spct}, \texttt{clear.spct}, \texttt{clear_body.spct}, \texttt{filter_cps.mspct}, \texttt{green_leaf.spct}, \texttt{opaque.spct}, \texttt{phenylalanine.spct}, \texttt{photodiode.spct}, \texttt{polyester.spct}, \texttt{sun.daily.data}, \texttt{sun.daily.spct}, \texttt{sun.data}, \texttt{sun.spct}, \texttt{water.spct}, \texttt{white_body.spct}, \texttt{white_led.cps_spct}, \texttt{white_led.raw_spct}, \texttt{white_led.source_spct}, \texttt{yellow_gel.spct}

**Examples**

\texttt{Ler_leaf.spct}

---

\texttt{Ler_leaf_rflt.spct} \hspace{1cm} \textit{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**

\texttt{Ler_leaf_rflt.spct}

**Format**

An \texttt{reflector_spct} object with 1750 rows and 2 variables
Ler_leaf_trns.spct

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  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)
Ler_leaf_trns_i.spct

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

Examples
Ler_leaf_trns_i.spct

---

### Logarithms and Exponentials

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))

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`, `^.generic_spct()`, `convolve_each()`, `div-.generic_spct`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`

---

MathFun

Miscellaneous Mathematical Functions

Description

abs\( (x) \) computes the absolute value of \( x \), sqrt\( (x) \) computes the (principal) square root of \( x \). The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of \( x \) and the current value of output options.

Usage

```r
## S3 method for class 'generic_spct'
sqrt(x)

## S3 method for class 'generic_spct'
abs(x)
```

Arguments

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

See Also

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

Merge a filter_spct with a reflector_spct returning an object_spct object, even if wavelength values are mismatched.

Usage

merge2object_spct(
  x,
  y,
  by = "w.length",
  ...,
  w.length.out = x["w.length"],
  Tfr.type.out = "total"
)

Arguments

x, y  a filter_spct object and a reflector_spct object.
by  a vector of shared column names in x and y to merge on; by defaults to w.length.
...  other arguments passed to dplyr::inner_join().
w.length.out  numeric vector of wavelengths to be used for the returned object (nm).
Tfr.type.out  character string indicating whether transmittance values in the returned object should be expressed as "total" or "internal". This applies only to the case when an object_spct is returned.

Value

An object_spct is returned as the result of merging a filter_spct and a reflector_spct object.

Note

If a numeric vector is supplied as argument for w.length.out, the two spectra are interpolated to the new wavelength values before merging. The default argument for w.length.out is x["w.length"].

See Also

join
merge_attributes

Merge and copy attributes

Description

Merge attributes from \textit{x} and \textit{y} and copy them to \textit{z}. Methods defined for spectral objects of classes from package 'photobiology'.

Usage

\texttt{merge\_attributes(x, y, z, which, which\_not, ...)}

\texttt{## Default S3 method:}
\texttt{merge\_attributes(x, y, z, which = NULL, which\_not = NULL, ...)}

\texttt{## S3 method for class 'generic\_spct'}
\texttt{merge\_attributes(}
\texttt{  x,}
\texttt{  y,}
\texttt{  z,}
\texttt{  which = NULL,}
\texttt{  which\_not = NULL,}
\texttt{  copy\_class = FALSE,}
\texttt{  ...}
\texttt{)}

Arguments

\texttt{x, y, z} \hspace{1cm} R objects. Objects \texttt{x} and \texttt{y} must be of the same class, \texttt{z} must be an object with a structure valid for this same class.

\texttt{which} \hspace{1cm} character Names of attributes to copy, if NULL all those relevant according to the class of \texttt{x} are used as default.

\texttt{which\_not} \hspace{1cm} character Names of attributes not to be copied. The names passed here are removed from the list for \texttt{which}, which is most useful when we want to modify the default.

\texttt{...} \hspace{1cm} not used

\texttt{copy\_class} \hspace{1cm} logical If TRUE class attributes are also copied.

Value

A copy of \texttt{z} with additional attributes set.

Methods (by class)

- \texttt{merge\_attributes(default)}: Default for generic function
- \texttt{merge\_attributes(generic\_spct)}:
minus-.generic_spct  Arithmetic Operators

Description
Subtraction operator for generic spectra.

Usage
## S3 method for class 'generic_spct'
e1 - e2 = NULL

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

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

mod-.generic_spct  Arithmetic Operators

Description
Reminder operator for generic spectra.

Usage
## S3 method for class 'generic_spct'
e1 %% e2

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

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

Multi-spct transform methods

Description

Apply a function or operator to a collection of spectra.

Usage

msmsply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)

msdply(
  mspct,
  .fun,
  ...,  
  idx = NULL,
  col.names = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

mslply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)

msaply(mspct, .fun, ..., .drop = TRUE, .parallel = FALSE, .paropts = NULL)

Arguments

mspc an object of class generic_mspct or a derived class
.fun a function
... other arguments passed to .fun
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
.idx character Name of the column with the names of the members of the collection of spectra.
col.names character Names to be used for data columns.
.drop should extra dimensions of length 1 in the output be dropped, simplifying the output. Defaults to TRUE
Value

a collection of spectra in the case of `msmsply`, belonging to a different class than `mspt` if `.fun` modifies the class of the member spectra.

a data frame in the case of `msdply`

a list in the case of `mslply`

a vector in the case of `msaply`

---

`mspct_classes` Names of multi-spectra classes

Description

Function that returns a vector containing the names of multi-spectra classes using for collections of spectra.

Usage

`mspct_classes()`

Value

A character vector of class names.

Examples

`mspct_classes()`

---

`na.omit` Handle Missing Values in Objects

Description

These methods are useful for dealing with NAs in e.g., `source_spct`, `response_spct`, `filter_spct` and `reflector_spct`.
Usage

## S3 method for class 'generic_spct'
na.omit(object, na.action = "omit", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'response_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'filter_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'reflector_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'object_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'solute_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_spct'
na.exclude(object, na.action = "exclude", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'response_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'filter_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'reflector_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'object_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'solute_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)
## S3 method for class 'object_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'solute_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

### Arguments
- **object**: an R object
- **na.action**: character One of "omit", "exclude" or "replace".
- **fill**: numeric Value used to replace NAs unless NULL, in which case interpolation is attempted.
- **target.colnames**: character Vector of names for the target columns to operate upon, if present in object.
- **...**: further arguments other special methods could require

### Details
If `na.omit` removes cases, the row numbers of the cases form the "na.action" attribute of the result, of class "omit".

`na.exclude` differs from `na.omit` only in the class of the "na.action" attribute of the result, which is "exclude".

### Note
`na.fail` and `na.pass` do not require a specialisation for spectral objects. R’s definitions work as expected with no need to override them. We do not define a method `na.replace`, just pass "replace" as argument. The current implementation replaces by interpolation only individual NAs which are flanked on both sides by valid data. Runs of multiple NAs can only replaced by a constant value passed through parameter `fill`.

### See Also
- `na.fail` and `na.action`
**Examples**

```r
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)
```

---

**net_irradiance**

*Net radiation flux*

**Description**

Estimate net radiation balance expressed as a flux in W/m². If `lw.down.irradiance` is passed a value in W / m² the difference is computed directly and if not an approximate value is estimated, using $R_{rel} = 0.75$ which corresponds to clear sky, i.e., uncorrected for cloudiness. This is the approach to estimation is that recommended by FAO for hourly estimates while here we use it for instantaneous or mean flux rates.

**Usage**

```r
net_irradiance(
  temperature,
  sw.down.irradiance,
  lw.down.irradiance = NULL,
  sw.albedo = 0.23,
  lw.emissivity = 0.98,
  water.vp = 0,
  R_rel = 1
)
```
Arguments

- `temperature` numeric vector of air temperatures (C) at 2 m height.
- `sw.down.irradiance, lw.down.irradiance` numeric Down-welling short wave and long wave radiation radiation (W/m2).
- `sw.albedo` numeric Albedo as a fraction of one (/1).
- `lw.emissivity` numeric Emissivity of the surface (ground or vegetation) for long wave radiation.
- `water.vp` numeric vector of water vapour pressure in air (Pa), ignored if `lw.down.irradiance` is available.
- `R_rel` numeric The ratio of actual and clear sky short wave irradiance (/1).

Value

A numeric vector of evapotranspiration estimates expressed as W / m-2.

See Also

Other Evapotranspiration and energy balance related functions: `ET_ref()`

---

**normalization**  

Normalization of an R object

Description

Normalization wavelength [nm] 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

Value

A single numeric value of wavelength [nm].
**normalize**

**Methods (by class)**

- normalization(default): Default methods.
- normalization(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**

```r
normalize(x, ...)

normalise(x, ...)
```

## Default S3 method:

```r
normalize(x, ...)
```

## S3 method for class 'source_spct'

```r
normalize(
  x,
  ...,
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  keep.scaling = FALSE,
  na.rm = FALSE
)
```

## S3 method for class 'response_spct'

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

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

## S3 method for class 'reflector_spct'

```r
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  qty.out = NULL,
  keep.scaling = FALSE,
  na.rm = FALSE
)
```

## S3 method for class 'solute_spct'

```r
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE
)
```

## S3 method for class 'raw_spct'

```r
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  keep.scaling = FALSE,
  na.rm = FALSE
)
```

## S3 method for class 'cps_spct'

```r
normalize(
  x,
  ..., 
  range = NULL,
```
normalize

```r
norm = "max",
keep.scaling = FALSE,
na.rm = FALSE
)

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

## S3 method for class 'source_mspct'
normalize(
  x,
  ..., 
  range = NULL,
norm = "max",
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
keep.scaling = FALSE,
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,
  ..., 
  range = NULL,
norm = "max",
qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
```
Arguments

x                   An R object
...                 not used in current version
range               An R object on which range() returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm) used to set boundaries for search for normalization.
norm                numeric Normalization wavelength (nm) or character string "max", or "min" for normalization at the corresponding wavelength, "update" to update the normal-
Normalization after modifying units of expression, quantity or range but respecting the previously used criterion, or "skip" to force return of x unchanged.

- **unit.out** character Allowed values "energy", and "photon", or its alias "quantum"
- **keep.scaling** logical Flag to indicate if any existing scaling should be preserved or not. The default, FALSE, preserves the behaviour of versions (<= 0.10.9).
- **na.rm** logical indicating whether 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 the minimum. It is also possible to update an existing normalization for different units of expression or after a conversion to a related spectral quantity.

By default the function is applied to the whole spectrum, but by passing a range of wavelengths as input, the search can be limited to a region of interest within the spectrum.

In 'photobiology' (>= 0.10.8) detailed information about the normalization is stored in an attribute. In 'photobiology' (>= 0.10.10) applying a new normalization to an already normalized spectrum recomputes the multiplier factors stored in the attributes whenever possible. This ensures that the returned object is identical independently of the previous application of a different normalization.

**Value**

A copy of x, with spectral data values normalized to one for the criterion specified by the argument passed to norm with information about the normalization applied saved in attributes "normalized" and "normalization".

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. Attributes normalized and normalization are set to keep a log of the computations applied.

**Methods (by class)**

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

Note

If the spectrum passed as argument to x has been previously scaled, in 'photobiology' (<= 0.10.9) the scaling attribute was always removed and no normalization factors returned. In 'photobiology' (>= 0.10.10) scaling information can be preserved by passing keep.scaling = TRUE (experimental feature).

normalize() 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 striping 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)
normalize(sun.spct) # equivalent

normalize(sun.spct, norm = "max")
normalize(sun.spct, norm = 400)
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:
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)

## S3 method for class 'generic_spct'
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)

## S3 method for class 'generic_mspct'
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)

- `normalized_diff_ind(default)`: default
- `normalized_diff_ind(generic_spct)`: 
- `normalized_diff_ind(generic_mspct)`: 

Note

Some NDxI indexes are directly based on satellite instrument data, such as those in the Landsat satellites. To simulate such indexes using spectral 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

`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.
opaque.spct

Value

a numeric vector of length two, guaranteed not to have missing values.

Examples

normalize_range_arg(c(NA, 500), range(sun.spct))
normalize_range_arg(c(300, NA), range(sun.spct))
normalize_range_arg(c(100, 5000), range(sun.spct), FALSE)
normalize_range_arg(c(NA, NA), range(sun.spct))
normalize_range_arg(c(NA, NA), sun.spct)

opaque.spct  Theoretical spectrum of an opaque material

Description

A dataset for a hypothetical object with transmittance 0/1 (0%)
**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
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()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

**Examples**

```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**

**Peaks or local maxima**

**Description**

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
peaks(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)
```

## S3 method for class 'numeric'

```r
```
peaks(x, span = 5, ignore_threshold = NA, strict = TRUE, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
peaks(
  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'
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,
  ...
```r
peaks

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 'solute_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,
```

peaks

    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",
    ...
    .parallel = FALSE,
    .paropts = NULL
)

## 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",
peaks(...,
  .parallel = FALSE,
  .paropts = NULL
)

## 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",
  ...
  .parallel = FALSE,
  .paropts = NULL
)

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

## 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",
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## 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
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.

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.

Logical If TRUE, an element must be strictly greater than all other values in its window to be considered a peak.

Logical indicating whether NA values should be stripped before searching for peaks.

Character Name of column where to look for peaks.

Logical Flag indicating if peak location should be refined by fitting a function.

Character String with the name of a method. Currently only spline interpolation is implemented.

Character One of "energy" or "photon"

Character One of "transmittance" or "absorbance"

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.

A subset of x with rows corresponding to local maxima.

**Methods (by class)**

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

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

---

### phenylalanine.spct

**Molar spectral attenuation coefficient of phenylalanine**

**Description**

A dataset containing the wavelengths at a 0.25 nm interval and the corresponding attenuation coefficients.

**Usage**

`phenylalanine.spct`

**Format**

A solute_spct object with 1993 rows and 2 variables
Details

- w.length (nm), range 222 to 720 nm.
- K.mole (cm-1/M)

Author(s)
Du et al. (original data); Scott Prahl (included data).

References

https://omlc.org/spectra/PhotochemCAD/html/073.html


See Also


Examples

head(phenylalanine.spct)
summary(phenylalanine.spct)
solute_properties(phenylalanine.spct)
cat(comment(phenylalanine.spct))

photodiode.spct

Spectral response of a GaAsP photodiode

Description

A dataset containing wavelengths at a 1 nm interval and spectral response as $A/(W/nm)$ for GaAsP photodiode type G6262 from Hamamatsu. Data digitized from manufacturer's data sheet. The value at the peak is 0.19 $A/W$.

Usage

photodiode.spct
Format

A response_spct object with 94 rows and 2 variables

Details

• w.length (nm).
• s.e.response (A/W)

References


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)
)
**photon_irradiance**

**Arguments**

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

**Value**

A single numeric value giving the ratio moles-photons per Joule.

**Note**

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

**See Also**

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

**Examples**

```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 in $[W m^{-2} nm^{-1}]$ or $[mol s^{-1} sm^{-2} nm^{-1}]$ as indicated by the argument passed to unit.in.

w.band waveband.

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

check.spectrum logical Flag telling whether to sanity check input data, default is TRUE.

use.cached.mult logical Flag telling whether multiplier values should be cached between calls.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value with no change in scale factor: $[mol s^{-1} sm^{-2}]$.

See Also

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

Examples

with(sun.data, photon_irradiance(w.length, s.e.irrad))
with(sun.data, photon_irradiance(w.length, s.e.irrad, new_waveband(400,700)))
photons_ratio

Description
This function gives the photon ratio between two given wavebands of a radiation spectrum.

Usage

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

Arguments

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

Value

  a single numeric value giving the unitless ratio.

Note

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

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

Examples

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

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'
polyester.spct

Usage
## S3 method for class 'generic_mspct'
polyester.spct
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 usegetOption("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)

- print(generic_mspct):

Note

This is simply a wrapper on the print method for tibbles, with additional information in the header. Currently, width applies only to the table of data.

Examples

```r
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

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

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

Arguments

x  an R object
...

Note

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

See Also

Other Local solar time functions: `as.solar_date()`, `is.solar_time()`, `solar_time()`

Description

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

Usage

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

Arguments

x  An object of one of the summary classes for spectra
...

Examples

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

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 Time of day functions: as_tod(), format.tod_time()

print.waveband

Description
A function to more nicely print objects of class "waveband".

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

Arguments
x an object of class "waveband"
... not used in current version
prod_spectra

Multiply two spectra, even if the wavelengths values differ

Description

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

Usage

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

Arguments

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

Details

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

Value

  a dataframe with two numeric variables

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

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)

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()
qe_ratio

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)

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

---

qe_ratio

*Photon:energy ratio*

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

## S3 method for class 'source_spct'
qe_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
)```
## S3 method for class 'source_mspct'

### qe_ratio

```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]", ""),
  ...
)
```

**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 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)

• qe_ratio(default): Default for generic function
• qe_ratio(source_spct): Method for source_spct objects
• qe_ratio(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.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other photon and energy ratio functions: e_ratio(), eq_ratio(), q_ratio()

Examples

qe_ratio(sun.spct, new_waveband(400,700))
Photon fluence

**Description**

Photon irradiance (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'

```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",
  ..., attr2tb = NULL, idx = "spct.idx",
  .parallel = FALSE, .paropts = NULL
)
```

### Arguments

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

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

Methods (by class)

• q_fluence(default): Default for generic function
• q_fluence(source_spct): Calculate photon fluence from a source_spct object and the duration of the exposure
• q_fluence(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 wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_fluence(), e_irrad(), fluence(), irrad(), q_irrad()

Examples

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

q_irrad \hspace{1cm} \textit{Photon irradiance}

Description

Photon irradiance (i.e. quantum irradiance) for one or more wavebands of a light source spectrum.
q_irrad

Usage

q_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

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

## S3 method for class 'source_spct'
q_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
q_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  ...)
time.unit = NULL,
scale.factor = 1,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = NULL,
allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...
attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
.paropts = NULL
)

Arguments

spct an R object.
w.band a list of waveband objects or a waveband object.
quantity character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
time.unit character or lubridate::duration object.
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
use.cached.mult logical indicating whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
... other arguments (possibly ignored).
naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
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, \([\text{W m}^{-2} \text{ nm}^{-1}] \rightarrow [\text{mol s}^{-1} \text{ m}^{-2}]\) If `time.unit` is day, \([\text{J d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}] \rightarrow [\text{mol d}^{-1} \text{ m}^{-2}]\)

Methods (by class)

- `q_irrad` (default): Default for generic function
- `q_irrad(source_spct)`: Calculates photon irradiance from a `source_spct` object.
- `q_irrad(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()`

Examples

```r
q_irrad(sun.spct, waveband(c(400,700)))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")
```
Description

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

Usage

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

## Default S3 method:

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

## S3 method for class `source_spct`

```r
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`

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

- q_ratio(default): Default for generic function
- q_ratio(source_spct): Method for source_spct objects
- q_ratio(source_mspct): Calculates photon:photon from a source_mspct object.

Note

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

See Also

Other photon and energy ratio functions: e_ratio(), eq_ratio(), qe_ratio()

Examples

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

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

## S3 method for class 'response_spct'
q_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

## S3 method for class 'response_mspct'
q_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)
```
Arguments

spct  an R object.

w.band  waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.

quantity  character string One of "total", "average", or "mean", "contribution", "contribution.pc", "relative", or "relative.pc".

time.unit  character or lubridate::duration object.

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

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

use.hinges  logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

...  other arguments (possibly used by derived methods).

naming  character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb  character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx  character Name of the column with the names of the members of the collection of spectra.

.parallel  if TRUE, apply function in parallel, using parallel backend provided by foreach

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

Value

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

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

- `q_response` (default): Default method for generic function
- `q_response(response_spct)`: Method for response spectra.
- `q_response(response_mspct)`: Calculates photon (quantum) response from a `response_mspct`

Note

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

See Also

Other response functions: `e_response()`, `response()`

Examples

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

---

**r4p_pkgs**

*Packages in R for Photobiology suite*

**Description**

A dataset containing the names of all the packages in this suite.

**Usage**

```r
r4p_pkgs
```

**Format**

A character vector.

**Details**

A character vector.

**Examples**

```r
r4p_pkgs
```
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

```r
# default, adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct))
spct
class(spct)

# adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'spct.idx' with the names given to the spectra in the list # supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'ID' with the names given to the spectra in the list # supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct),
                   idfactor = "ID")
head(spct)
class(spct)
```
Description

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

Usage

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

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

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

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

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

### 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 if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.

### Value

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

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

Methods (by class)

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

Note

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

Examples

```r
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 approxima-
tion 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 \texttt{NA} might be useful in some cases.

References

Examples

\begin{verbatim}
relative_AM(c(90, 60, 30, 1, -10))
relative_AM(c(90, 60, 30, 1, -10), occluded.value = Inf)
relative_AM(zenith.angle = 0)
\end{verbatim}

replace_bad_pixs  \hspace{1cm} \textit{Replace bad pixels in a spectrum}

Description
This function replaces data for bad pixels by a local estimate, by either simple interpolation or using the algorithm of Whitaker and Hayes (2018).

Usage

\begin{verbatim}
replace_bad_pixs(
  x,
  bad.pix.idx = FALSE,
  window.width = 11,
  method = "run.mean",
  na.rm = TRUE
)
\end{verbatim}

Arguments
\begin{itemize}
  \item \texttt{x} \hspace{1cm} numeric vector containing spectral data.
  \item \texttt{bad.pix.idx} \hspace{1cm} logical vector or integer. Index into bad pixels in \texttt{x}.
  \item \texttt{window.width} \hspace{1cm} integer. The full width of the window used for the running mean.
\end{itemize}
replace_bad_pixs

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
# 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 or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- **unit.out**: character, Allowed values "energy", and "photon", or its alias "quantum".
- **quantity**: character string, One of "average" or "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".
- **time.unit**: character or lubridate::duration object.
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical, if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.hinges**: logical, Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments (possibly used by derived methods).
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
- **idx**: character, Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts

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

Value

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

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

Methods (by class)

- `response` (default): Default for generic function
- `response(response_spct)`: Method for response spectra.
- `response(response_mspct)`: Calculates response from a response_mspct

Note

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

See Also

Other response functions: `e_response()`, `q_response()`
**Usage**

- `Rfr_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5, p_fraction = 0.5)`
- `Rfr_p_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)`
- `Rfr_s_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)`

**Arguments**

- `angle_deg, angle`  
  numeric vector Angle of incidence of the light beam, in degrees or radians. If both are supplied, radians take precedence.
- `n`  
  numeric vector, or `generic_spct` object Relative refractive index. The default 1.5 is suitable for crown glass or acrylic interacting with visible light. `n` depends on wavelength, more or less strongly depending on the material.
- `p_fraction`  
  numeric in range 0 to 1. Polarization, defaults to 0.5 assuming light that is not polarized.

**Details**

These functions implement Fresnel’s formulae. All parameters accept vectors as arguments. If both `n` and `angle` are vectors with length different from one, they should both have the same length. Reflectance depends on polarization, the `s` and `p` components need to be computed separately and added up. `Rfr_from_n()` is for non-polarized light, i.e., with equal contribution of the two components.

**Value**

If `n` is a numeric vector the returned value is a vector of reflectances, while if `n` is a `generic_spct` object the returned value is a `reflector_spct` object.

**Examples**

- `Rfr_from_n(0:90)`
- `Rfr_from_n(0:90, p_fraction = 1)`
- `Rfr_from_n(0:90, n = 1.333) # water`

**rgb_spct**

<table>
<thead>
<tr>
<th>Description</th>
<th>RGB color values</th>
</tr>
</thead>
</table>

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)

References

For more information on the usage of rmDerivedMspct, please refer to the documentation provided with the package.

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.
rmDerivedSpct

Remove "generic_spct" and derived class attributes.

Description

Removes from a spectrum object the class attributes "generic_spct" and any derived class attribute such as "source_spct". **This operation is done by reference!**

Usage

rmDerivedSpct(x, keep.classes = NULL)

Arguments

- **x**: an R object.
- **keep.classes**: character vector. Names of classes to keep. Can be used to retain base class "generic_spct".

Details

This function alters x itself by reference. If x is not a generic_spct object, x is not modified. This function behaves similarly to setdiff() but preserving the original order of the character vector of the S3 class names.

Value

A character vector containing the removed class attribute values. This is different to the behaviour of function unlist in base R!

Note

If x is an object of any of the spectral classes defined in this package, this function changes by reference the spectrum object into the underlying data.frame object. Otherwise, it just leaves x unchanged.

See Also

Other set and unset spectral class functions: setGenericSpct()
### Examples

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

### round

**Rounding of Numbers**

**Description**

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`. `floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`. `trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward 0. `round` rounds the values in its first argument to the specified number of decimal places (default 0). `signif` rounds the values in its first argument to the specified number of significant digits. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of `x` and the current value of output options.

**Usage**

```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.
select_spct_attributes

Merge user supplied attribute names with default ones

Description

Allow users to add and subtract from default attributes in addition to providing a given set of attributes.

Usage

select_spct_attributes(attributes, attributes.default = spct_attributes())

spct_attributes(.class = "all", attributes = ")

Arguments

attributes, attributes.default
  character vector or a list of character vectors.

.class
  character Name of spectral class.

Details

Vectors of character strings passed as argument to attributes are parsed so that if the first member string is "+", the remaining members are added to those in attributes.default; if it is "-" the remaining members are removed from in attributes.default; and if it is "=" the remaining members replace those in in attributes.default. If the first member is none of these three strings, the behaviour is the same as when the first string is "=". If attributes is NULL all the attributes in attributes.default are used and if it is "" no attribute names are returned, "" has precedence over other member values. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

A character vector of attribute names.
setBSWFUsed

See Also

get_attributes

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getSoluteProperties(), getWhatMeasured(),
getWhereMeasured(), get_attributes(), isValidInstrDesc(), isValidInstrSettings(), setFilterProperties(),
setWhereMeasured(), spct_attr2tb(), spct_metadata(), trimInstrDesc(),
trimInstrSettings()

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 cur-
rently not fully supported.

See Also

Other BSWF attribute functions: getBSWFUsed()
Description

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

Usage

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

```r
filter_properties(x) <- value
```

Arguments

- `x`: a filter_spct object
- `filter.properties, value`: a list with fields named "Rfr.constant", "thickness" and "attenuation.mode".
- `pass.null`: logical If TRUE, the parameters to the next three parameters will be always ignored, otherwise they will be used to build an object of class "filter.properties" when the argument passed to parameter `filter.properties` is NULL.
- `Rfr.constant`: numeric The value of the reflection factor [\(1\)].
- `thickness`: numeric The thickness of the material [\(m\)].
- `attenuation.mode`: character One of "reflection", "absorption", "absorption.layer" 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 \texttt{filter_spct} object, \( x \) is not modified.

The values of \texttt{attenuation.mode} "reflection" and "absorption" should be used when one of these processes is clearly the main one; "mixed" is for cases when they both play a role, i.e., when a simple correction using a single value of \( R_{fr} \) across wavelengths is not possible; "absorption.layer" is for cases when a thin absorbing layer is deposited on the surface of a transparent support or enclosed between two sheets of glass or other transparent material. If in doubt, set this to \texttt{NA} to ensure that computation of spectra for a different thickness remains disabled.

See Also

Other measurement metadata functions: \texttt{add_attr2tb()}, \texttt{getFilterProperties()}, \texttt{getHowMeasured()}, \texttt{getInstrDesc()}, \texttt{getInstrSettings()}, \texttt{getSoluteProperties()}, \texttt{getWhatMeasured()}, \texttt{getWhenMeasured()}, \texttt{getWhereMeasured()}, \texttt{get\_attributes()}, \texttt{isValidInstrDesc()}, \texttt{isValidInstrSettings()}, \texttt{select\_spct\_attributes()}, \texttt{setHowMeasured()}, \texttt{setInstrDesc()}, \texttt{setInstrSettings()}, \texttt{setSoluteProperties()}, \texttt{setWhatMeasured()}, \texttt{setWhenMeasured()}, \texttt{setWhereMeasured()}, \texttt{spct\_attr2tb()}, \texttt{spct\_metadata()}, \texttt{trimInstrDesc()}, \texttt{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)
```

---

**setGenericSpct**

*Convert an R object into a spectrum object.*

**Description**

Sets the class attribute of a data.frame or an object of a derived class to "generic\_spct".

**Usage**

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

multiple.wl = 1L,
idfactor = NULL
)

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

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

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

setSoluteSpct(
  x,
  K.type = c("attenuation", "absorption", "scattering"),
  name = NA_character_,
  mass = NA_character_,
  formula = NA_character_,
  structure = grDevices::as.raster(matrix()),
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

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

Arguments

- **x**: data.frame, list or generic_spct and derived classes
- **multiple.wl**: numeric Maximum number of repeated w.length entries with same value.
- **idfactor**: character Name of factor distinguishing multiple spectra when stored longitudinally (required if multiple.wl > 1).
- **strict.range**: logical Flag indicating whether off-range values result in an error instead of a warning.
- **time.unit**: character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.
- **Tfr.type**: character Either "total" or "internal".
- **Rfr.constant**: numeric The value of the reflection factor [1/1].
- **thickness**: numeric The thickness of the material.
attenuation.mode
character One of "reflection", "absorption" or "mixed".

K.type
character A string, either "attenuation", "absorption" or "scattering".

name, solvent.name
character The names of the substance and of the solvent. A named character
vector, with member names such as "IUPAC" for the authority.

mass
numeric The mass in Dalton (Da = g/mol).

formula
character The molecular formula.

structure
raster A bitmap of the structure.

ID, solvent.ID
character The IDs of the substance and of the solvent. A named character
vector, with member names such as "ChemSpider" or "PubChen" for the authority.

Rfr.type
character A string, either "total" or "specular".

response.type
a character string, either "response" or "action".

bswf.used
character A string, either "none" or the name of a BSWF. (Users seldom need
to change the default, as this metadata value is in normal use set by operators or
functions that apply a BSWF.)

Details
This method alters x itself by reference and in addition returns the modified x invisibly. The wave-
length values and data are checked for validity and out-of-range values trigger warnings. These
checks are done during construction by means of the matching check_spct methods, unless checks
have been disabled by setting the corresponding option (see enable_check_spct).

Value
x

Functions

- setCalibrationSpct(): Set class of a an object to "calibration_spct".
- setRawSpct(): Set class of a an object to "raw_spct".
- setCpsSpct(): Set class of a an object to "cps_spct".
- setFilterSpct(): Set class of an object to "filter_spct".
- setSoluteSpct(): Set class of an object to "solute_spct".
- setReflectorSpct(): Set class of a an object to "reflector_spct".
- setObjectSpct(): Set class of an object to "object_spct".
- setResponseSpct(): Set class of an object to "response_spct".
- setSourceSpct(): Set class of an object to "source_spct".
- setChromaSpct(): Set class of an object to "chroma_spct".

Warning!
Not entering metadata when creating an object will limit the available operations!
Note

"internal" transmittance is defined as the transmittance of the material body itself, while "total" transmittance includes the effects of surface reflectance on the amount of light transmitted. For non-diffusing materials like glass an approximate \( R_{fr}.constant \) value can be used to inter-convert total and internal transmittance values. Use NA if the mode is not known, or not applicable, e.g., for materials subject to internal scattering. The validity of computations related to thickness of the material or length of the light path depends on the availability and accuracy of the metadata.

Particles in suspension unlike dissolved solutes scatter light. Thus two different processes can attenuate light in liquid media: absorption and scattering. Coefficients of attenuation are always based on measurements of internal absorbance or internal transmittance. In practice this is achieved by using as reference pure solvent in a vessel, such as a spectrometer cuvette, called blank. The measurement of the blank is done sequentially, before or after the sample of interest in single beam spectrophotometers and concurrently in double beam spectrophotometers. \( K.type \) describes the process of attenuation: "attenuation", "absorption" or "scattering", with "attenuation" used for cases of mixed modes of attenuation. Set \( K.type = NA \) if not available or unknown, or not applicable.

"specular" reflectance is defined as that measured by collecting the light reflected by the surface at the "mirror" of the angle of incidence; i.e., using a probe with a narrow angle of aperture. Usually measured close to normal angle of incidence. "total" reflectance is defined as that measured by collecting all the light reflected by the surface; i.e., using an integrating sphere. In a mirror, reflectance is mostly specular, while on the white surface of a sheet of paper scattering predominates. In the first case the value for total reflectance is not much more than for specular reflectance, while in the second case the difference is much larger as the "specular" component is much smaller.

See Also

Other set and unset spectral class functions: \texttt{rmDerivedSpct()}

Examples

\begin{verbatim}
my.df <- data.frame(w.length = 300:309, s.e.irrad = rep(100, 10))
is.source_spct(my.df)
setSourceSpct(my.df)
is.source_spct(my.df)
\end{verbatim}

---

**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.
Set the "instr.desc" attribute

Usage

setInstrDesc(x, instr.desc)

Arguments

x          a generic_spct object

instr.desc  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.

The fields to be passed in the list instr.desc in part vary depending on the instrument brand and model.
setInstrSettings

Set the "instr.settings" attribute

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

**setKType**

*Set the "K.type" attribute*

**Description**

Function to set by reference the "K.type" attribute of an existing solute_spct object.

**Usage**

```
setKType(x, K.type = c("attenuation", "absorption", "scattering"))
```

**Arguments**

- `x`: a solute_spct or a summary_solute_spct object.
- `K.type`: character A string, either "attenuation", "absorption" or "scattering".

**Value**

`x`

**Note**

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a solute_spct object, `x` is not modified. The behaviour of this function is 'unusual' in that the default for parameter `K.type` is used only if `x` does not already have this attribute set.

**See Also**

Other K attribute functions: `getKType()`

**Examples**

````
print("missing example")
```
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" and "normalization" attributes*

Description

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

Usage

```r
setNormalized(
x,  
norm = FALSE,
norm.type = NA_character_,  
norm.factors = NA_real_,  
norm.cols = NA_character_,  
norm.range = rep(NA_real_, 2)
)
```

```r
setNormalised(
x,  
norm = FALSE,
norm.type = NA_character_,  
norm.factors = NA_real_,  
norm.cols = NA_character_,  
norm.range = rep(NA_real_, 2)
)
```
Arguments

- `x`: a generic_spect object.
- `norm`: numeric (or logical) Normalization wavelength (nanometres).
- `norm.type`: character Type of normalization applied.
- `norm.factors`: numeric The scaling factor(s) so that dividing the spectral values by this factor reverts the normalization.
- `norm.cols`: character The name(s) of the data columns normalized.
- `norm.range`: numeric The wavelength range used for normalization (nm).

Note

If `x` is not a generic_spect object, `x` is not modified. Passing a logical as argument to `norm` is deprecated but kept for backwards compatibility.

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

See Also

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

---

`setResponseType`  
**Set the "response.type" attribute**

Description

Function to set by reference the "response.type" attribute of an existing response_spect object.

Usage

```r
setResponseType(x, response.type = c("response", "action"))
```

Arguments

- `x`: a response_spect object
- `response.type`: a character string, either "response" or "action"

Details

Objects of class `response_spect()` can contain data for a response spectrum or an action spectrum. Response spectra are measured using the same photon (or energy) irradiance at each wavelength. Action spectra are derived from dose response curves at each wavelength, and responsivity at each wavelength is expressed as the reciprocal of the photon fluence required to obtain a fixed level of response.
setRfrType

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a response_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter response.type is used only if x does not already have this attribute set.

Examples

my.spct <- ccd.spct
setResponseType(my.spct, "action")

Description

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

Usage

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

Arguments

x a reflector_spct or an object_spct 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_spct or object_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter Rfr.type is used only if x does not already have this attribute set.

See Also

Other Rfr attribute functions: getRfrType()
Examples
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
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.
setSoluteProperties

Methods (by class)

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

Note

if x is not a generic_spct object, x is not modified.

See Also

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

setSoluteProperties  Set the "solute.properties" attribute

Description

Function to set by reference the "solute.properties" attribute of an existing solute_spct object.

Usage

setSoluteProperties(
  x,
  solute.properties = NULL,
  pass.null = FALSE,
  mass = NA_real_,
  formula = NULL,
  structure = grDevices::as.raster(matrix()),
  name = NA_character_,
  ID = NA_character_,
  solvent.name = NA_character_,
  solvent.ID = NA_character_,
)

solute_properties(x) <- value

Arguments

x

solute_spct A spectrum of coefficients of attenuation.
solute.properties, value

a list with fields named "mass", "formula", "structure", "name" and "ID".
pass.null  logical If TRUE, the parameters to the next three parameters will be always ignored, otherwise they will be used to build an object of class "solute.properties" when the argument to solute.properties is NULL.

mass numeric The mass in Dalton \([Da = g \text{ mol}^{-1}]\).

formula character The molecular formula.

structure raster A bitmap of the structure.

name, solvent.name character The name of the substance and the name of the solvent. A named character vector, with member names such as "IUPAC" for the authority.

ID, solvent.ID character The names of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChen" for the authority.

Details

Storing solute properties allows inter-conversion between bases of expression, and ensures the unambiguous identification of the substances to which the spectral data refer. These properties make it possible to compute filter_spect objects for solutions of the solute, i.e., absorption spectra of liquid filters. The parameter pass.null makes it possible to remove the attribute. The solvent used for the determination of the attenuation coefficient is important metadata as the solvent can alter the spectral absorption properties of the solute.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a filter_spect object, x is not modified.

See Also


Examples

solute.properties <-
list(formula = c(text = "H2O", html = "H<sub>2</sub>\), TeX = "$H_2O$"),
  name = c("water", IUPAC = "oxidane"),
  structure = grDevices::as.raster(matrix()),
  mass = 18.015, # Da
  ID = c(ChemSpider = "917", CID = "962"),
solvent.name = NA_character_,
solvent.ID = NA_character_)
my.spct <- solute_spct()
solute_properties(my.spct) <- solute.properties
solute_properties(my.spct)
solute_properties(my.spct) <- NULL
solute_properties(my.spct)
solute_properties(my.spct, return.null = TRUE)
solute_properties(my.spct)

setTfrType

setTfrType

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

Usage
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()

Examples
my.spct <- polyester.spct
getTfrType(my.spct)
setTfrType(my.spct, "internal")
getTfrType(my.spct)
setTimeUnit

Set the "time.unit" attribute of an existing source_spct object

Description

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

Usage

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

Arguments

  x  
  a source_spct object

  time.unit  
  character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.

  override.ok  
  logical Flag that can be used to silence warning when overwriting an existing attribute value (used internally)

Value

  x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a source_spct or response_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter time.unit is used only if x does not already have this attribute set. time.unit = "hour" is currently not fully supported.

See Also

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

Examples

my.spct <- sun.spct
setTimeUnit(my.spct, time.unit = "second")
setTimeUnit(my.spct, time.unit = lubridate::duration(1, "seconds"))
setWhatMeasured

Set the "what.measured" attribute

Description

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

Usage

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

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
```

```r
## Default S3 method:
setWhenMeasured(x, when.measured, ...)
```

```r
## S3 method for class 'generic_spct'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)
```

```r
## S3 method for class 'summary_generic_spct'
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)
```

```r
## 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)**

- setWhenMeasured(default): default
- setWhenMeasured(generic_spct): generic_spct
- setWhenMeasured(summary_generic_spct): summary_generic_spct
- setWhenMeasured(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 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

```r
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

```r
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**, **value**: 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)

- `setWhereMeasured(default)`: default
- `setWhereMeasured(generic_spct)`: generic_spct
- `setWhereMeasured(summary_generic_spct)`: summary_generic_spct
- `setWhereMeasured(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)
where_measured(my.spct) <- data.frame(lon = 0, lat = -60)
where_measured(my.spct)
```
shared_member_class  Classes common to all collection members.

Description
Find the set intersection among the class attributes of all collection member as a target set of class names.

Usage
shared_member_class(l, target.set = spct_classes())

Arguments
1  a list or a generic_mspct object or of a derived class.
target.set  character The target set of classes within which to search for classes common to all members.

Value
A character vector containing the class attribute values.

See Also
Other set and unset 'multi spectral' class functions: rmDerivedMspct()

sign  Sign

Description
sign returns a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).

Usage
## S3 method for class 'generic_spct'
sign(x)

Arguments
x  an object of class "generic_spct"

See Also
Other math operators and functions: MathFun,^.generic_spct(),convolve_each(),div-.generic_spct,log(),minus-.generic_spct,mod-.generic_spct,plus-.generic_spct,round(),slash-.generic_spct,times-.generic_spct
### 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, wl.range, ...)
```

### Examples
```r
## Default S3 method:
smooth_spct(x, method, strength, wl.range, ...)

## S3 method for class 'source_spct'
smooth_spct(
x,
method = "custom",
strength = 1,
wl.range = NULL,
)```
```r
na.rm = FALSE,
...
)

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

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

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

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

## S3 method for class 'generic_mspct'
smooth_spct(
  x,
  method = "custom",
```
smooth_spct

```r
strength = 1,
wl.range = NULL,
na.rm = FALSE,
...
)
```

Arguments

- **x**: an R object.
- **method**: a character string "custom", "lowess", "supsmu" or "skip".
- **strength**: numeric value to adjust the degree of smoothing. Ignored if method-specific parameters are passed through ....
- **wl.range**: any R object on which applying the method range() yields a vector of two numeric values, describing a range of wavelengths (nm) within which spectral data is to be smoothed. NA is interpreted as the min or max value of x[[w.length]].
- **...**: other parameters passed to the underlying smoothing functions.
- **na.rm**: logical A flag indicating whether NA values should be stripped before the computation proceeds.

Value

A copy of x with spectral data values replaced by smoothed ones.

Methods (by class)

- smooth_spct(default): Default for generic function
- smooth_spct(source_spct): Smooth a source spectrum
- smooth_spct(filter_spct): Smooth a filter spectrum
- smooth_spct(reflector_spct): Smooth a reflector spectrum
- smooth_spct(solute_spct): Smooth a solute attenuation spectrum
- smooth_spct(response_spct): Smooth a response spectrum
- smooth_spct(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 case of methods "lowess" and "supsmu" the current function behaves like a wrapper of the functions of the same names from base R. Method "skip" returns x unchanged.
Examples

```r
my.spct <- clip_wl(sun.spct, c(400, 500))
smooth_spct(my.spct)
smooth_spct(my.spct, method = "custom", strength = 1)
smooth_spct(my.spct, method = "custom", strength = 4)
smooth_spct(my.spct, method = "supsmu", strength = 4)
```

solar_time

### Local solar time

Description

`solar_time()` computes the time of day expressed in seconds since the astronomical midnight using and instant in time and a geocode as input. Solar time is useful when we want to plot data according to the local solar time rather than the local time in use at a time zone. How the returned instant in time is expressed depends on the argument passed to `unit.out`.

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", "time", "hour", "minute", or "second".

Details

Solar time is determined by the position of the sun in the sky and it almost always differs from the time expressed in the local time coordinates in use. The differences can vary from a few minutes up to a couple of hours depending on the exact location within the time zone and the use or not of daylight saving time.

Value

In all cases solar time is expressed as time since local astronomical midnight and, thus, lacks date information. If `unit.out = "time"`, a numeric value in seconds with an additional class attribute "solar_time"; if `unit.out = "datetime"`, a "POSIXct" value in seconds from midnight but with an additional class attribute "solar_date"; if `unit.out = "hour"` or `unit.out = "minute"` or `unit.out = "second"`, a numeric value.
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

as_tod

Other Local solar time functions: as.solar_date(), is.solar_time(), print.solar_time()

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)

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

Description

These constructor functions can be used to create spectral objects derived from generic_spct. They take as arguments numeric vectors for the wavelengths and spectral data, and numeric, character, and logical values for metadata attributes to be saved to the objects created and options controlling the creation process.
Usage

source_spct(
  w.length = NULL,
  s.e.irrad = NULL,
  s.q.irrad = NULL,
  ...,  
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

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

raw_spct(
  w.length = NULL,
  counts = NA_real_,
  ...,  
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL
)

cps_spct(
  w.length = NULL,
  cps = NA_real_,
  ...,  
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL
)

generic_spct(
  w.length = NULL,
source_spct

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

response_spct(
  w.length = NULL,
  s.e.response = NULL,
  s.q.response = NULL,
  ...
  time.unit = c("second", "day", "exposure"),
  response.type = c("response", "action"),
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL
)

filter_spct(
  w.length = NULL,
  Tfr = NULL,
  Tpc = NULL,
  Afr = NULL,
  A = NULL,
  ...
  Tfr.type = c("total", "internal"),
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA,
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

reflector_spct(
  w.length = NULL,
  Rfr = NULL,
  Rpc = NULL,
  ...
  Rfr.type = c("total", "specular"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

solute_spct(
w.length = NULL,
K.mole = NULL,
K.mass = NULL,
attenuation.XS = NULL,
..., 
log.base = 10,
K.type = c("attenuation", "absorption", "scattering"),
name = NA_character_,
mass = NA_character_,
formula = NULL,
structure = grDevices::as.raster(matrix()),
ID = NA_character_,
solvent.name = NA_character_,
solvent.ID = NA_character_,
comment = NULL,
strict.range =getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL
)

object_spct(
  w.length = NULL,
  Rfr = NULL,
  Tfr = NULL,
  Afr = NULL,
  ..., 
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  comment = NULL,
  strict.range =getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

chroma_spct(
  w.length = NULL,
  x,
  y,
  z,
  ..., 
  comment = NULL,
  strict.range =getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

**Arguments**

- **w.length** numeric vector with wavelengths in nanometres [nm].
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.e.irrad</td>
<td>numeric vector with spectral energy irradiance in ([W \text{ m}^{-2} \text{ nm}^{-1}]) or ([J \text{ d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]).</td>
</tr>
<tr>
<td>s.q.irrad</td>
<td>numeric A vector with spectral photon irradiance in ([mol \text{ s}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]) or ([mol \text{ d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]).</td>
</tr>
<tr>
<td>...</td>
<td>other arguments passed to <code>tibble()</code> such as vectors or factors to be added as additional columns.</td>
</tr>
<tr>
<td>time.unit</td>
<td>character string indicating the time unit used for spectral irradiance or exposure (&quot;second&quot;, &quot;day&quot; or &quot;exposure&quot;) or an object of class duration as defined in package lubridate.</td>
</tr>
<tr>
<td>bswf.used</td>
<td>character A string indicating the BSWF used, if any, for spectral effective irradiance or exposure (&quot;none&quot; or the name of the BSWF).</td>
</tr>
<tr>
<td>comment</td>
<td>character A string to be added as a comment attribute to the object created.</td>
</tr>
<tr>
<td>strict.range</td>
<td>logical Flag indicating whether off-range values result in an error instead of a warning.</td>
</tr>
<tr>
<td>multiple.wl</td>
<td>numeric Maximum number of repeated (w.length) entries with same value. (As with multiple spectra stored in long from).</td>
</tr>
<tr>
<td>idfactor</td>
<td>character Name of factor distinguishing multiple spectra when stored longitudinally (required if (multiple.wl &gt; 1)).</td>
</tr>
<tr>
<td>irrad.mult</td>
<td>numeric vector with multipliers for each detector pixel expressed in units of (W \text{ m}^{-2} \text{ nm}^{-1} \text{ m}^{-1} \text{s}), where (n \text{ s}^{-1}) are detector counts per second.</td>
</tr>
<tr>
<td>instr.desc</td>
<td>a list describing the spectrometer used to acquire the data.</td>
</tr>
<tr>
<td>counts</td>
<td>numeric vector with raw counts expressed per scan.</td>
</tr>
<tr>
<td>instr.settings</td>
<td>a list describing the settings used to acquire the data.</td>
</tr>
<tr>
<td>cps</td>
<td>numeric vector with linearized raw counts expressed per second (n \text{ s}^{-1}).</td>
</tr>
<tr>
<td>s.e.response</td>
<td>numeric vector with a biological, chemical or physical response expressed per unit spectral energy irradiance ([W \text{ m}^{-2} \text{ nm}^{-1} \text{ m}^{-1} \text{s}^{-1}]).</td>
</tr>
<tr>
<td>s.q.response</td>
<td>numeric vector with a biological, chemical or physical response expressed per unit spectral photon irradiance in ([mol \text{ s}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]) or ([mol \text{ d}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]).</td>
</tr>
<tr>
<td>response.type</td>
<td>a character string, either &quot;response&quot; or &quot;action&quot;.</td>
</tr>
<tr>
<td>Tfr</td>
<td>numeric vector with spectral transmittance as fraction of one ([/1]).</td>
</tr>
<tr>
<td>Tpc</td>
<td>numeric vector with spectral transmittance as percent values.</td>
</tr>
<tr>
<td>Afr</td>
<td>numeric vector of absorptance as fraction of one ([/1]).</td>
</tr>
<tr>
<td>A</td>
<td>numeric vector of absorbance values ((\log_{10}\text{-base a.u.})).</td>
</tr>
<tr>
<td>Tfr.type</td>
<td>character string indicating whether transmittance and absorptance values are &quot;total&quot; or &quot;internal&quot; values.</td>
</tr>
<tr>
<td>Rfr.constant</td>
<td>numeric The value of the reflection factor ([/1]).</td>
</tr>
<tr>
<td>thickness</td>
<td>numeric The thickness of the material.</td>
</tr>
<tr>
<td>attenuation.mode</td>
<td>character One of &quot;reflection&quot;, &quot;absorption&quot; or &quot;mixed&quot;.</td>
</tr>
<tr>
<td>Rfr</td>
<td>numeric vector with spectral reflectance as fraction of one ([/1]).</td>
</tr>
<tr>
<td>Rpc</td>
<td>numeric vector with spectral reflectance as percent values.</td>
</tr>
</tbody>
</table>
Rfr.type character A string, either "total" or "specular".
K.mole numeric vector with molar attenuation coefficient in SI units $[m^2 mol^{-1}]$.
K.mass numeric vector with mass attenuation coefficient in SI units $[m^2 g^{-1}]$.
attenuation.XS numeric vector with attenuation cross section values (Converted during object construction into K.mole.)
log.base numeric Normally one of e or 10. Data are stored always on base 10 corresponding to decadal absorbance as used in chemistry.
K.type character A string, either "attenuation", "absorption" or "scattering".
nname, solvent.name character The names of the substance and of the solvent. A named character vector, with member names such as "IUPAC" for the authority.
mass numeric The molar mass in Dalton [Da] ($Da = g mol^{-1}$).
formula character The molecular formula.
structure raster A bitmap of the structure.
ID, solvent.ID character The ID of the substance and of the solvent. A named character vector, with member names such as "ChemSpider" or "PubChem" for the authority.
x, y, z numeric colour coordinates

Details
Constructors can be used to create spectral objects from spectral quantities expressed on a single base or unit. Some of the functions have different formal parameters accepting a quantity expressed in different units, however, an argument can be passed to only one of these formal parameters in a given call. The constructors object_spct() and chroma_spct() require arguments to be passed for multiple but distinct spectral quantities.

Value
A object of class generic_spct or a class derived from it, depending on the function used. In other words an object of a class with the same name as the constructor function.

Warning for filter_spct!
Not entering metadata when creating an object will limit the available operations! While "internal" transmittance is defined as the transmittance of the material body itself, "total" transmittance includes the effects of surface reflectance on the amount of light transmitted. For non-diffusing materials like glass an approximate Rfr.constant value can be used to convert "total" into "internal" transmittance values and vice versa. Use NA if not known, or not applicable, e.g., for materials subject to internal scattering.

Warning for solute_spct!
You should always set the base for logarithms to match that on which the absorbance data are expressed. Failing to do this will result in bad data and all further computation will be wrong. Not entering metadata when creating an object will limit the available operations! Mass should be indicated in daltons or $g mol^{-1}$. The SI unit of molar attenuation coefficient is the square metre.
per mole ($m^2 mol^{-1}$), but in practice, quantities are usually expressed in terms of $M^{-1} cm^{-1}$ or $l mol^{-1} cm^{-1}$ (the latter two units are both equal to 0.1 $m^2 mol^{-1}$ and quantities expressed in them need to be divided by 10 when passed as arguments to K.mole.).

See Also

setFilterProperties
setSoluteProperties

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

---

**spct_attr2tb**

**Copy attributes into a tibble**

**Description**

Method returning attributes of an object of class generic_spct or derived, or of class waveband. Only attributes defined and/or set by package 'photobiology' for objects of the corresponding class are returned.

**Usage**

```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`.

See Also

**spct_classes**

*Function returning a vector containing the names of spectra classes.*

**Description**

Function returning a vector containing the names of spectra classes.

**Usage**

```r
spct_classes()
```

**Value**

A character vector of class names.

**Examples**

```r
spct_classes()
```

---

**spct_metadata**

*Access metadata*

**Description**

Return metadata attributes from a single spectrum or a collection of spectra as a tibble.

**Usage**

```r
spct_metadata(
  x,
  col.names = NULL,
  idx = "spct.idx",
  na.rm = is.null(col.names),
  unnest = TRUE
)
```

**Arguments**

- `x` generic_mspct or generic_spct Any collection of spectra or spectrum.
- `col.names` named character vector Name(s) of column(s) to create.
- `idx` character Name of the column with the names of the members of the collection of spectra.
- `na.rm` logical Flag controlling deletion of columns containing only NA values.
- `unnest` logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.
Details

Attributes are returned as columns in a tibble. If the argument to `col.names` is a 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.

Some metadata values are stored in lists or data frames, these can be returned as a list columns or the individual fields unnested into separate columns.

Value

A tibble With the metadata attributes and an index column.

See Also

`add_attr2tb` for more details.


Examples

```r
my.mspct <- source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))
spct_metadata(my.mspct)
spct_metadata(sun.spct)
spct_metadata(my.mspct, na.rm = TRUE)
spct_metadata(sun.spct, na.rm = TRUE)
spct_metadata(my.mspct, col.names = c(geocode = "geo", "instr.desc"))
spct_metadata(sun.spct, col.names = c(geocode = "geo", "instr.desc"))
spct_metadata(sun.spct, col.names = "where.measured")$where.measured
```
Description

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

Usage

spikes(x, z.threshold, max.spike.width, na.rm, ...)

## Default S3 method:
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'numeric'
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

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

## S3 method for class 'generic_spct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
...,
var.name = NULL
)

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

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

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

## S3 method for class 'reflector_spct'
spikes(x, z.threshold = 9, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'solute_spct'
spikes(x, z.threshold = 9, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'cps_spct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
...,
var.name = "cps"
)

## S3 method for class 'raw_spct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
...,
var.name = "counts"
## S3 method for class 'generic_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
  var.name = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

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

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

## S3 method for class 'filter_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  filter_qty = getOption("photobiology.filterqty", default = "transmittance"),
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
## S3 method for class 'reflector_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'solute_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., var.name = "cps",
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'raw_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., var.name = "counts",
  .parallel = FALSE,
  .paropts = NULL
)
Arguments

- **x**: an R object
- **z.threshold**: numeric, Modified Z values larger than `z.threshold` are considered to correspond to spikes.
- **max.spike.width**: integer, wider regions with high Z values are not detected as spikes.
- **na.rm**: logical indicating whether NA values should be stripped before searching for spikes.
- **...**: ignored
- **var.name, y.var.name**: character, Name of column where to look for spikes.
- **unit.out**: character, One of "energy" or "photon"
- **filter.qty**: character, One of "transmittance" or "absorbance"
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A subset of `x` with rows corresponding to spikes.

Methods (by class)

- **spikes** (default): Default returning always NA.
- **spikes** (numeric): Default function usable on numeric vectors.
- **spikes** (data.frame): Method for "data.frame" objects.
- **spikes** (generic_spect): Method for "generic_spect" objects.
- **spikes** (source_spect): Method for "source_spect" objects.
- **spikes** (filter_spect): Method for "filter_spect" objects.
- **spikes** (reflector_spect): Method for "reflector_spect" objects.
- **spikes** (solute_spect): Method for "solute_spect" objects.
- **spikes** (cps_spect): Method for "cps_spect" objects.
- **spikes** (generic_mspct): Method for "generic_mspct" objects.
- **spikes** (source_mspct): Method for "source_mspct" objects.
- **spikes** (response_mspct): Method for "cps_mspct" objects.
- **spikes** (filter_mspct): Method for "filter_mspct" objects.
- **spikes** (reflector_mspct): Method for "reflector_mspct" objects.
- **spikes** (solute_mspct): Method for "solute_mspct" objects.
- **spikes** (cps_mspct): Method for "cps_mspct" objects.
See Also

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

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

Examples

```r
spikes(sun.spct)
```

split2mspct  

Convert a 'wide' or untidy data frame into a collection of spectra

Description

Convert a data frame object into a "multi spectrum" object by constructing a an object of a multi-spct class, converting numeric columns other than wavelength into individual spct objects.

Usage

```r
split2mspct(
  x,
  member.class = NULL,
  spct.data.var = NULL,
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
```

```r
split2source_mspct(
  x,
  spct.data.var = "s.e.irrad",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
```

```r
split2response_mspct(
  x,
  spct.data.var = "s.e.response",
  w.length.var = "w.length",
  idx.var = NULL,
```
\begin{verbatim}
ncol = 1,
byrow = FALSE,
...
)
split2filter_mspct(
  x,
  spct.data.var = "Tfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
split2reflector_mspct(
  x,
  spct.data.var = "Rfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
split2solute_mspct(
  x,
  spct.data.var = "K.mole",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
split2cps_mspct(
  x,
  spct.data.var = "cps",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)
split2raw_mspct(
  x,
  spct.data.var = "count",
\end{verbatim}
split_bands

w.length.var = "w.length",
idx.var = NULL,
ncol = 1,
byrow = FALSE,
...
)
split2calibration_mspct(
  x,
  spct.data.var = "irrad.mult",
w.length.var = "w.length",
idx.var = NULL,
ncol = 1,
byrow = FALSE,
...
)

Arguments

x               data frame
member.class    character Class of the collection members
spct.data.var   character Name of the spectral data argument in the object constructor for member.class
w.length.var    character Name of column containing wavelength data in nanometres
idx.var         character Name of column containing data to be copied unchanged to each spct object
ncol            integer Number of 'virtual' columns in data
byrow           logical If ncol > 1 how to read in the data
...             additional named arguments passed to the member constructor function.

See Also

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

split_bands List-of-wavebands constructor

Description

Build a list of unweighted "waveband" objects that can be used as input when calculating irradiances.
split_bands

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_spect 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 them-
  selves 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.spect, length.out=10)
split_bands(waveband(c(400,700)), length.out=5)
split_energy_irradiance

Energy irradiance for split spectrum regions

Description

This function returns the energy irradiance for a series of contiguous wavebands from a radiation-source spectrum. The returned values can be either absolute or relative to their sum.

Usage

split_energy_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)

Arguments

w.length numeric vector of wavelengths (nm).
s.irrad numeric vector of spectral (energy or photon) irradiance values (W m-2 nm-1) or (mol s-1 m-2 nm-1).
cut.w.length numeric vector of wavelengths (nm).
unit.in character string with allowed values "energy", and "photon", or its alias "quantum".
scale character string indicating the scale used for the returned values ("absolute", "relative", "percent").
check.spectrum logical indicating whether to sanity check input data, default is TRUE.
use.cached.mult logical Flag indicating whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a numeric vector of irradiances with no change in scale factor: [W m-2 nm-1] -> [W m-2] or [mol s-1 m-2] -> [W m-2] or relative values (fraction of one) if scale = "relative" or scale = "percent".
Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

See Also

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

Examples

```r
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)
)
```
split_irradiance

Arguments

- **w.length**: numeric Vector of wavelengths [nm].
- **s.irrad**: numeric vector of spectral irradiances in \([W\ m^{-2}\ nm^{-1}]\) or \([mol\ s^{-1}\ sm^{-2}\ nm^{-1}]\) as indicated by the argument passed to unit.in.
- **cut.w.length**: numeric Vector of wavelengths [nm].
- **unit.out**, **unit.in**: character Allowed values "energy", and "photon", or its alias "quantum".
- **scale**: a character A string indicating the scale used for the returned values ("absolute", "relative" or "percent").
- **check.spectrum**: logical Flag indicating whether to sanity check input data, default is TRUE.
- **use.cached.mult**: logical Flag indicating whether multiplier values should be cached between calls.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A numeric vector of irradiances with no change in scale factor if scale == "absolute", \([W\ m^{-2}]\) or \([mol\ s^{-1}\ sm^{-2}]\) depending on the argument passed to unit.out or relative values (as fraction of one if scale == "relative" or percentages if scale == "percent" of photons or energy depending on the argument passed to unit.out.

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set check.spectrum=FALSE then you should call check_spectrum at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

Examples

```r
with(sun.data,

  split_irradiance(w.length, s.e.irrad,
                    cut.w.length = c(300, 400, 500, 600, 700),
                    unit.out = "photon"))
```
**split_photon_irradiance**

*Photon irradiance for split spectrum regions*

**Description**

This function returns the photon irradiance for a series of contiguous wavebands from a radiation spectrum. The returned values can be either absolute or relative to their sum.

**Usage**

```r
split_photon_irradiance(
  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 = range(w.length),  # numeric vector of wavelengths (nm).
  unit.in = "energy",  # character Allowed values "energy", and "photon", or its alias "quantum".
  scale = "absolute",  # a character A string indicating the scale used for the returned values ("absolute", "relative", "percent").
  check.spectrum = TRUE,  # logical Flag indicating whether to sanity check input data, default is TRUE.
  use.cached.mult = FALSE,  # logical Flag indicating whether multiplier values should be cached between calls.
  use.hinges = getOption("photobiology.use.hinges", default = NULL)  # 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.
)
```

**Arguments**

- **w.length**: numeric vector of wavelengths (nm).
- **s.irrad**: numeric vector of spectral (energy or photon) irradiance values (W m-2 nm-1).
- **cut.w.length**: numeric vector of wavelengths (nm).
- **unit.in**: character Allowed values "energy", and "photon", or its alias "quantum".
- **scale**: a character A string indicating the scale used for the returned values ("absolute", "relative", "percent").
- **check.spectrum**: logical Flag indicating whether to sanity check input data, default is TRUE.
- **use.cached.mult**: logical Flag indicating whether multiplier values should be cached between calls.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

**Value**

a numeric vector of photon irradiances with no change in scale factor: [W m-2 nm-1] -> [mol s-1 m-2], [mol s-1 m-2 nm-1] -> [mol s-1 m-2] or relative values (fraction of one based on photon units) if scale = "relative" or scale = "percent".
spread

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set check.spectrum=FALSE then you should call check_spectrum at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

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

Examples

with(sun.data,
    split_photon_irradiance(w.length, s.e.irrad,
        cut.w.length = c(300, 400, 500, 600, 700)))
with(sun.data,
    split_photon_irradiance(w.length, s.e.irrad))

spread

Expanse

Description

A method that returns the expanse \((\max(x) - \min(x))\) for R objects. In particular the wavelength \([\text{nm}]\) expanse of the wavelength range of objects of classes waveband or of class generic_spct or derived (or the expanse of values in a numeric vector).

Usage

spread(x, ...)

wl_expanse(x, ...)

expanse(x, ...)

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

## S3 method for class 'numeric'
expanse(x, ...)
## S3 method for class 'waveband'
expanse(x, ...)

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

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

**Arguments**

- **x**: an R object
- **...**: not used in current version
- **idx**: character Name of the column with the names of the members of the collection of spectra.

**Value**

A numeric value equal to \( \max(x) - \min(x) \). In the case of spectral objects wavelength difference \([nm]\). For any other R object, according to available specialised methods of **min** and **max**.

**Methods (by class)**

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

**Examples**

expanse(10:20)
expanse(sun.spct)
wl_expanse(sun.spct)
expanse(sun.spct)

---

**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 `[` indexing operator.
- **...**: further arguments to be passed to or from other methods.

Value

An object similar to `x` containing just the selected rows and columns. Depending on the columns remaining after subsetting the class of the object will be simplified to the most derived parent class.

Note

This method is copied from `base::subset.data.frame()` but ensures that all metadata stored in attributes of spectral objects are copied to the returned value.

Examples

```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 spectra object of the corresponding class. For data frames converting numeric columns other than wavelength into individual `spct` objects.

Usage

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

See Also

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

---

**subt_spectra**  
*Subtract two spectra*

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added. This is 'parallel' operation between two spectra.
subt_spectra

Usage

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

Arguments

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

Details

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

Value

a data frame with two numeric variables

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

### 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
)
```

**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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.length</td>
<td>A numeric vector with the wavelengths (nm) obtained by &quot;fusing&quot; w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.</td>
</tr>
<tr>
<td>s.irrad</td>
<td>A numeric vector with the sum of the two spectral values at each wavelength.</td>
</tr>
</tbody>
</table>

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

```r
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
sun.daily.spct

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

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
sun.spct

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

table(sun.spct)

---

sun.spct Solar spectral irradiance (simulated)

Description

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance and spectral photon irradiance. Values simulated for 22 June 2010, near midday, at Helsinki, under partly cloudy conditions. The variables are as follows:

Usage

sun.spct

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

Description

Function sun_angles() returns the solar angles and Sun to Earth relative distance for given times and locations using a very precise algorithm. Convenience functions sun_azimuth(), sun_elevation(), sun_zenith_angle() and distance_to_sun() are wrappers on sun_angles() that return individual vectors.

Usage

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()

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
)

distance_to_sun(
  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. The function is vectorized and in
particular passing a vector of times for a single geocode enhances performance very much as the
equation of time, the most time consuming step, is computed only once.
For improved performance, if more than one angle is needed it is preferable to directly call `sun_angles`
instead of the wrapper functions as this avoids the unnecessary recalculation.

**Value**

A data.frame with variables time (in same TZ as input), TZ, solartime, longitude, latitude,
address, azimuth, elevation, declination, eq.of.time, hour.angle, and distance. 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 by `sun_angles`. Angles
are expressed in degrees, solartime is a vector of class "solar.time", distance is expressed in
relative sun units.

**Important!**

Given an instant in time and a time zone, the date is computed from these, and may differ by one day
to that at the location pointed by geocode at the same instant in time, unless the argument passed
to tz matches the time zone at this location.

**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**

The primary source for the algorithm used is the book: Meeus, J. (1998) Astronomical Algorithms,

A different implementation is available at https://apps-nefsc.fisheries.noaa.gov/AstroCalc4R/.
An interactive web page using the same algorithms is available at https://gml.noaa.gov/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()`

**Examples**

```r
library(lubridate)
sun_angles()
sun_azimuth()
sun_elevation()
sun_zenith_angle()
sun_angles(ymd_hms("2014-09-23 12:00:00"))
```
s_e_irrad2rgb

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))

---

**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**

```r
s_e_irrad2rgb(
  w.length, # numeric vector of wavelengths (nm).
  s.e.irrad, # numeric vector of spectral irradiance values.
  sens = photobiology::ciexyzCMF2.spct, # 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 = NULL, # character string for naming the rgb color definition.
  check = TRUE # logical indicating whether to check or not spectral data.
)
```

**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 string for naming the rgb color definition.

**Value**

A color defined using `rgb`. The numeric values of the RGB components can be obtained using function `col2rgb`.

**Note**

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


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

```r
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

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

## Default S3 method:

```r
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

## S3 method for class 'source_mspct'

```r
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

## S3 method for class 'response_mspct'

```r
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

## S3 method for class 'filter_mspct'

```r
s_mean(x, trim = 0, na.rm = FALSE, ...)
```
s_mean

```r
## S3 method for class 'reflector_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)
```

### Arguments

- **x** An R object 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_spct", containing the mean spectrum.

### Methods (by class)

- s_mean(default):
- s_mean(source_mspct):
- s_mean(response_mspct):
- s_mean(filter_mspct):
- s_mean(reflector_mspct):
- s_mean(calibration_mspct):
- s_mean(cps_mspct):
- s_mean(raw_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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.
s_mean_se

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

s_mean_se(x, na.rm, mult, ...)

## Default S3 method:
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'filter_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'source_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'response_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'reflector_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'calibration_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'cps_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

## S3 method for class 'raw_mspct'
S3method(x, na.rm = FALSE, mult = 1, ...)

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.

See Also

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

- s_mean_se(default):
- s_mean_se(filter_mspct):
- s_mean_se(source_mspct):
- s_mean_se(response_mspct):
- s_mean_se(reflector_mspct):
- s_mean_se(calibration_mspct):
- s_mean_se(cps_mspct):
- s_mean_se(raw_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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

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

---

**s_mean_se_band**

*Mean plus and minus standard error from collection of spectra*

**Description**

A method to compute the mean of values and se 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**

```r
s_mean_se_band(x, na.rm, mult, ...)
```

### Default S3 method:
```
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)
```

### S3 method for class 'filter_mspct'
```
s_mean_se_band(x, na.rm = FALSE, mult = 1, ...)
```
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 of spectral objects, such as a "filter_mspct" object, the returned object is of the same class as the members of the collection, such as "filter_spct", containing the mean spectrum.

Methods (by class)

- `s_mean_se_band` (default):
- `s_mean_se_band` (filter_mspct):
- `s_mean_se_band` (source_mspct):
- `s_mean_se_band` (response_mspct):
- `s_mean_se_band` (reflector_mspct):
- `s_mean_se_band` (calibration_mspct):
- `s_mean_se_band` (cps_mspct):
- `s_mean_se_band` (raw_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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

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, ...)

## S3 method for class 'cps_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_median(x, na.rm = FALSE, ...)
Arguments

x  
An R object. 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)

- s_median(default):
- s_median(source_mspct):
- s_median(response_mspct):
- s_median(filter_mspct):
- s_median(reflector_mspct):
- s_median(calibration_mspct):
- s_median(cps_mspct):
- s_median(raw_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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

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

---

**s_prod**  
Product from collection of spectra

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.
**s_prod**

**Usage**

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

## Default S3 method:

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

## S3 method for class 'source_mspct'

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

## S3 method for class 'response_mspct'

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

## S3 method for class 'filter_mspct'

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

## S3 method for class 'reflector_mspct'

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

## S3 method for class 'calibration_mspct'

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

## S3 method for class 'cps_mspct'

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

## S3 method for class 'raw_mspct'

```r
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_spect", containing the product of the spectra.

**Methods (by class)**

- `s_prod(default)`
- `s_prod(source_mspct)`
- `s_prod(response_mspct)`
- `s_prod(filter_mspct)`
- `s_prod(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.

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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

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

s_range

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'
\texttt{s\_range(x, na.rm = FALSE, \ldots)}

\texttt{## S3 method for class 'cps\_mspct'}
\texttt{s\_range(x, na.rm = FALSE, \ldots)}

\texttt{## S3 method for class 'raw\_mspct'}
\texttt{s\_range(x, na.rm = FALSE, \ldots)}

**Arguments**

\textbf{x} \hspace{1cm} An R object. Currently this package defines methods for collections of spectral objects.

\textbf{na.rm} \hspace{1cm} logical. A value indicating whether NA values should be stripped before the computation proceeds.

\textbf{\ldots} \hspace{1cm} Further arguments passed to or from other methods.

**Value**

If \texttt{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)**

- \texttt{s\_range(default)}:
- \texttt{s\_range(filter\_mspct)}:
- \texttt{s\_range(source\_mspct)}:
- \texttt{s\_range(response\_mspct)}:
- \texttt{s\_range(reflector\_mspct)}:
- \texttt{s\_range(calibration\_mspct)}:
- \texttt{s\_range(cps\_mspct)}:
- \texttt{s\_range(raw\_mspct)}:

**Note**

Trimming of extreme values and omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in \texttt{x} must share the same set of wavelengths.

Objects of classes \texttt{raw\_spct} and \texttt{cps\_spct} can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of \texttt{cps\_spct} members.

**See Also**

See \texttt{Extremes} details on the \texttt{min()} and \texttt{max()} methods used for the computations.
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 'reflector_mspct'
s_sd(x, na.rm = FALSE, ...)

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

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

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

Arguments

x
An R object. 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)**

- \( \text{s.sd}(\text{default}) \):
- \( \text{s.sd}(\text{filter_mspct}) \):
- \( \text{s.sd}(\text{source_mspct}) \):
- \( \text{s.sd}(\text{response_mspct}) \):
- \( \text{s.sd}(\text{reflector_mspct}) \):
- \( \text{s.sd}(\text{calibration_mspct}) \):
- \( \text{s.sd}(\text{cps_mspct}) \):
- \( \text{s.sd}(\text{raw_mspct}) \):

**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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

**See Also**

See \( \text{sd} \) for details about \( \text{sd}() \) methods for other classes.

---

<table>
<thead>
<tr>
<th><strong>s.se</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Error of a collection of spectra</strong></td>
</tr>
</tbody>
</table>

**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**

\[
\text{s.se}(x, \text{na.rm}, \ldots)
\]

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

  ## S3 method for class 'source_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_se(x, na.rm = FALSE, ...)

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

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

## S3 method for class 'cps_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_se(x, na.rm = FALSE, ...)

Arguments

- **x**: An R object. 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)

- `s_se(default)`:  
- `s_se(source_mspct)`:  
- `s_se(response_mspct)`:  
- `s_se(filter_mspct)`:  
- `s_se(reflector_mspct)`:  
- `s_se(calibration_mspct)`:  
- `s_se(cps_mspct)`:  
- `s_se(raw_mspct)`:  

**s_sum**

**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.

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of `cps_spct` members.

---

**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:
```
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'filter_mspct'
```
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'source_mspct'
```
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'response_mspct'
```
s_sum(x, na.rm = FALSE, ...)
```

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

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

## S3 method for class 'cps_mspct'
```
s_sum(x, na.rm = FALSE, ...)
```

## S3 method for class 'raw_mspct'
```
s_sum(x, na.rm = FALSE, ...)
```
Arguments

- **x**: An R object. 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 sum of the spectra.

Methods (by class)

- `s_sum(default)`:  
- `s_sum(filter_mspct)`:  
- `s_sum(source_mspct)`:  
- `s_sum(response_mspct)`:  
- `s_sum(reflector_mspct)`:  
- `s_sum(calibration_mspct)`:  
- `s_sum(cps_mspct)`:  
- `s_sum(raw_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`.

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of `cps_spct` members.

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

**Arguments**

- **x**
  - An R object. 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)

- \( \text{s\_var}() \)
- \( \text{s\_var}(\text{filter\_mspct}) \)
- \( \text{s\_var}(\text{source\_mspct}) \)
- \( \text{s\_var}(\text{response\_mspct}) \)
- \( \text{s\_var}(\text{reflector\_mspct}) \)
- \( \text{s\_var}(\text{calibration\_mspct}) \)
- \( \text{s\_var}(\text{cps\_mspct}) \)
- \( \text{s\_var}(\text{raw\_mspct}) \)

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.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

See \( \text{cor} \) for details about \( \text{var}() \), which is used for the computations.

---

**T2A**

*Convert transmittance into absorbance.*

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,
  ...
)

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)

- T2A(default): Default method for generic function
- T2A(numeric): Method for numeric vectors
- T2A(filter_spct): Method for filter spectra
- T2A(filter_mspct): Method for collections of filter spectra

**See Also**

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

byref = FALSE,
clean = FALSE,
...
.parallel = FALSE,
.paropts = NULL
)

Arguments

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

Value

A copy of x with a column 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)

• T2Afr(default): Default method for generic function
• T2Afr(numeric): Default method for generic function
• T2Afr(filter_spct): Method for filter spectra
• T2Afr(object_spct): Method for object spectra
• T2Afr(filter_mspct): Method for collections of filter spectra
• T2Afr(object_mspct): Method for collections of object spectra

See Also

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

Examples

T2Afr(Ler_leaf.spct)
Description

Spectra are tagged by adding variables and attributes containing color definitions, labels, and a factor following the wavebands given in \textit{w.band}. This method is most useful for plotting realistic computed colors from spectral data.

Usage

tag(x, ...)  

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

## S3 method for class 'generic_spct'  
tag(  
  x,  
  w.band = NULL,  
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),  
  use.hinges = TRUE,  
  short.names = TRUE,  
  chroma.type = "CMF",  
  byref = FALSE,  
   ...)  

## S3 method for class 'generic_mspct'  
tag(  
  x,  
  w.band = NULL,  
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),  
  use.hinges = TRUE,  
  short.names = TRUE,  
  chroma.type = "CMF",  
  byref = FALSE,  
  ...,  
  .parallel = FALSE,  
  .paropts = NULL  
)

Arguments

\texttt{x} \hspace{1cm} an R object.  
\texttt{...} \hspace{1cm} ignored (possibly used by derived methods).
w.band waveband or list of waveband objects. The waveband(s) determine the region(s) of the spectrum that are tagged

wb.trim logical Flag telling if wavebands crossing spectral data boundaries are trimmed or ignored

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

short.names logical Flag indicating whether to use short or long names for wavebands

chroma.type character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system.

byref logical Flag indicating if new object will be created by reference or by copy of x

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

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

Value
A copy of x expanded with additional columns with color-related information.

Methods (by class)

- tag(default): Default method for generic
- tag(generic_spct): Tag one of generic_spct, and derived classes including source_spct, filter_spct, reflector_spct, object_spct, and response_spct.
- tag(generic_mspct): Tag one of generic_mspct, and derived classes including source_mspct, filter_mspct, reflector_mspct, object_mspct, and response_mspct.

Note
NULL as w.band argument does not add any new tags, instead it removes existing tags if present. NA, the default, as w.band argument removes existing waveband tags if present and sets the wl.color variable. If a waveband object or a list of wavebands is supplied as argument then tagging is based on them, and wl.color is also set.

See Also
Other tagging and related functions: is_tagged(), untag(), wb2rect_spct(), wb2spct(), wb2tagged_spct()

Examples

tag(sun.spct)
tag(sun.spct, list(A = waveband(c(300,3005))))
**thin_wl**

*Thin the density of wavelength values*

**Description**

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

**Usage**

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

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

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

### S3 method for class 'source_spect'
```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_spect'
```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_spect'
```r
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)
```

### S3 method for class 'reflector_spect'
```r
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, ...)
```
## S3 method for class 'solute_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, ...)

## S3 method for class 'raw_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'cps_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'object_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

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

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

## S3 method for class 'generic_mspct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, ...)

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

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

### Arguments

- **x** An R object
- **...** additional named arguments passed down to f.
- **max.wl.step** numeric. Largest allowed wavelength difference between adjacent spectral values in nanometres (nm).
- **max.slope.delta** numeric in 0 to 1. Largest allowed change in relative slope of the spectral quantity per nm between adjacent pairs of values.
- **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 where slope is shallow and featureless.

Methods (by class)

- \( \text{thin_wl}(\text{default}) \): Default for generic function
- \( \text{thin_wl}(\text{generic_spct}) \)
- \( \text{thin_wl}(\text{source_spct}) \)
- \( \text{thin_wl}(\text{response_spct}) \)
- \( \text{thin_wl}(\text{filter_spct}) \)
- \( \text{thin_wl}(\text{reflector_spct}) \)
- \( \text{thin_wl}(\text{solute_spct}) \)
- \( \text{thin_wl}(\text{raw_spct}) \)
- \( \text{thin_wl}(\text{cps_spct}) \)
- \( \text{thin_wl}(\text{object_spct}) \)
- \( \text{thin_wl}(\text{chroma_spct}) \)
- \( \text{thin_wl}(\text{calibration_spct}) \)
- \( \text{thin_wl}(\text{generic_mspct}) \)
- \( \text{thin_wl}(\text{chroma_mspct}) \)
- \( \text{thin_wl}(\text{calibration_mspct}) \)

Note

The value of \( \text{max.slope.delta} \) 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: \text{collect2mspct()}, \text{drop_user_cols()}, \text{uncollect2spct()}.

Examples

\[
\text{nrow(yellow_gel.spct)}
\]
\[
\text{wl_stepsizes(yellow_gel.spct)}
\]
\[
\text{thinned.spct} \leftarrow \text{thin_wl(yellow_gel.spct)}
\]
\[
\text{nrow(thinned.spct)}
\]
\[
\text{wl_stepsizes(thinned.spct)}
\]
Arithmetic Operators

Description
Multiplication operator for spectra.

Usage

## S3 method for class 'generic_spct'
e1 * e2

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

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

transmittance Transmittance

Description
Summary transmittance for supplied wavebands from filter or object spectrum.

Usage

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

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

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

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.
**transmittance**

1. **quantity**
   - character string. One of "average", "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".

2. **wb.trim**
   - logical. TRUE if wavebands crossing spectral data boundaries are trimmed, FALSE if they are discarded.

3. **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.

4. **...**
   - ignored (possibly used by derived methods).

5. **naming**
   - character. One of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

6. **attr2tb**
   - character vector. See [add_attr2tb](#) for the syntax for attr2tb passed as is to formal parameter col.names.

7. **idx**
   - character. Name of the column with the names of the members of the collection of spectra.

8. **.parallel**
   - if TRUE, apply function in parallel, using parallel backend provided by foreach.

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

### Value

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

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

### Methods (by class)

- transmittance(default): Default method
- transmittance(filter_spct): Method for filter spectra
- transmittance(object_spct): Method for object spectra
- transmittance(filter_mspct): Calculates transmittance from a filter_mspct
- transmittance(object_mspct): Calculates transmittance from an 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.
Trigonometric Functions

Description

Trigonometric functions for object of generic_spct and derived classes. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options.

Usage

```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)
```

Arguments

- `x`: an object of class "generic_spct" or a derived class.
trimInstrDesc

Trim the "instr.desc" attribute

Description

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

Usage

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

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**

```r
trim_spct(
  spct,
  range = NULL,
  low.limit = NULL,
  high.limit = NULL,
  use.hinges = TRUE,
  fill = NULL,
  byref = FALSE,
  verbose = getOption("photobiology.verbose")
)
```

```r
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
)
```

```r
trim2overlap(
  mspct,
  use.hinges = TRUE,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)
```

```r
extend2extremes(
  mspct,
  use.hinges = TRUE,
```
Arguments

- **spct**: an object of class "generic_spct".
- **range**: a numeric vector of length two, or any other object for which method range() will return a numeric vector of length two.
- **low.limit**: shortest wavelength to be kept (defaults to shortest w.length value).
- **high.limit**: longest wavelength to be kept (defaults to longest w.length value).
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **fill**: if fill==NULL then tails are deleted, otherwise tails or s.irrad are filled with the value of fill.
- **byref**: logical indicating if new object will be created by reference or by copy of spct.
- **verbose**: logical.
- **mspct**: an object of class "generic_mspct"
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach.
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

a spectrum 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**

**Examples**

```r
trim_spct(sun.spct, low.limit=300)
trim_spct(sun.spct, low.limit=300, fill=NULL)
trim_spct(sun.spct, low.limit=300, fill=NA)
trim_spct(sun.spct, low.limit=300, fill=0.0)
trim_spct(sun.spct, range = c(300, 400))
trim_spct(sun.spct, range = c(300, NA))
trim_spct(sun.spct, range = c(NA, 400))
```

**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**

```r
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.
trim_waveband

Value

A data.frame with variables x and y.

Note

When expanding a spectrum, if fill == NULL, expansion is not performed with a warning.

See Also

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

Examples

head(sun.data)
head(with(sun.data,
  trim_tails(w.length, s.e.irrad, low.limit=300)))
head(with(sun.data,
  trim_tails(w.length, s.e.irrad, low.limit=300, fill=NULL)))

trim_waveband

Trim (or expand) head and/or tail

Description

Trimming of waveband boundaries can be needed when the spectral data do not cover the whole waveband, or wavebands may have to be removed altogether.

Usage

trim_waveband(  
  w.band,  
  range = NULL,  
  low.limit = 0,  
  high.limit = Inf,  
  trim = getOption("photobiology.waveband.trim", default = TRUE),  
  use.hinges = TRUE,  
  trunc.labels = getOption("photobiology.brief.trunc.names", default = c("["", "]"))
)
trim_waveband

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 \( \text{trim} = \text{TRUE} \) is passed or excluded if \( \text{trim} = \text{FALSE} \). 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 \( w.\text{band} \). 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()`
 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),
  ...
)

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 = "!")
## S3 method for class 'list'
trim_wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  ...
)

### Arguments
- **x**: an R object.
- **range**: a numeric vector of length two, or any other object for which function range() will return two.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **fill**: if fill == NULL then tails are deleted, otherwise tails are filled with the value of fill.
- **...**: ignored (possibly used by derived methods).
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
- **trim**: logical (default is TRUE which trims the wavebands at the boundary, while FALSE discards wavebands that are partly off-boundary).

### Value
A copy of x, usually trimmed or expanded to a different length, either shorter or longer. Possibly with some of the original spectral data values replaced with fill.

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

Note

By default the \texttt{w.length} values for the first and last rows in the returned object are the values supplied as \texttt{range}.
trim\_wl when applied to waveband objects always inserts hinges when trimming.
trim\_wl when applied to waveband objects always inserts hinges when trimming.

See Also

Other trim functions: \texttt{clip\_wl()}, \texttt{trim\_spct()}, \texttt{trim\_waveband()}

Examples

\begin{verbatim}
trim\_wl(sun.spct, range = c(400, 500))
trim\_wl(sun.spct, range = c(NA, 500))
trim\_wl(sun.spct, range = c(400, NA))
\end{verbatim}

\begin{verbatim}
tz\_time\_diff
\end{verbatim}

\textbf{Time difference between two time zones}

Description

Returns the difference in local time expressed in hours between two time zones at a given instant in time. The difference due to daylight saving time or Summer and Winter time as well as historical changes in time zones are taken into account.

Usage

\begin{verbatim}
tz\_time\_diff(
    when = lubridate::now(),
    tz.target = lubridate::tz(when),
    tz.reference = "UTC"
)
\end{verbatim}

Arguments

\begin{verbatim}
when datetime A time instant
tz.target, tz.reference
    character Two time zones using names recognized by functions from package 'lubridate'
\end{verbatim}

Value

A numeric value.
Note

This function is implemented using functions from package 'lubridate'. For details on the handling of time zones, please, consult the documentation for `Sys.timezone` about system differences in time zone names and handling.

uncollect2spct Extract all members from a collection

Description

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

Usage

```r
uncollect2spct(x, ...)  
```

## Default S3 method:
```
uncollect2spct(x, ...)
```

## S3 method for class 'generic_mspct'
```
uncollect2spct(  
x,  
name.tag = ".spct",  
ignore.case = FALSE,  
check.names = TRUE,  
check.overwrite = TRUE,  
...  
)
```

Arguments

- **x** An R object
- **...** additional named arguments passed down to `f`.
- **name.tag** character. A string used as tag for the names of the objects. If of length zero, names of members are used as named of objects. Otherwise the tag is appended, unless already present in the member name.
- **ignore.case** logical. If FALSE, the pattern matching used for `name.tag` is case sensitive and if TRUE, case is ignored during matching.
- **check.names** logical. If TRUE then the names of the objects created are checked to ensure that they are syntactically valid variable names and unique. If necessary they are adjusted (by `make.names`) so that they are, and if FALSE names are used as is.
- **check.overwrite** logical. If TRUE trigger an error if an existing object would be overwritten, and if FALSE silently overwrite objects.
Value

Utility used for its side effects, invisibly returns a character vector with the names of the objects created.

Methods (by class)

- `uncollect2spct(default)`: Default for generic function
- `uncollect2spct(generic_mspct)`: 

See Also

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

Examples

```r
my.mspct <- source_mspct(list(sun1.spct = sun.spct, sun2.spct = sun.spct))
uncollect2spct(my.mspct)
ls(pattern = "*.spct")
```

Description

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

Usage

```r
untag(x, ...)
```

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

## S3 method for class 'generic_spct'
```r
untag(x, byref = FALSE, ...)
```

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

```r
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". `obj.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.
validate_geocode

Usage

using_Tfr(expr)
using_Afr(expr)
using_A(expr)
using_energy(expr)
using_photon(expr)
using_quantum(expr)

Arguments

expr      an R expression to execute.

Value

The value returned by the execution of expression.

References

Based on withOptions() as offered by Thomas Lumley, and listed in https://www.burns-stat.com/the-options-mechanism-in-r/, section Deep End, of "The Options mechanism in R" by Patrick Burns.

<table>
<thead>
<tr>
<th>validate_geocode</th>
<th>Validate a geocode</th>
</tr>
</thead>
</table>

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()

---

valleys

Valleys or local minima

Description

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

Usage

valleys(x, span, 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,
nr = 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,
nr = FALSE,
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'solute_spct'
valleys(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
nr = FALSE,
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'cps_spct'
valleys(
x,
span = 5,
ignore_threshold = 0,
valleys

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,
  refine.wl = FALSE,
  method = "spline",
  ...,
  parallel = FALSE,
  paropts = NULL
)

## S3 method for class 'solute_mspct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  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 compu-
tation 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)

- valleys(default): Default function usable on numeric vectors.
- valleys(default): Default returning always NA.
- valleys(numeric): Default function usable on numeric vectors.
- valleys(data.frame): Method for "data.frame" objects.
- valleys(generic_spct): Method for "generic_spct" objects.
- valleys(filter_spct): Method for "filter_spct" objects.
- valleys(reflector_spct): Method for "reflector_spct".
- valleys(cps_spct): Method for "cps_spct" objects.
- valleys(generic_mspct): Method for "generic_mspct" objects.
• valleys(response_mspct): Method for "cps_mspct" objects.
• valleys(filter_mspct): Method for "filter_mspct" objects.
• valleys(reflector_mspct): Method for "reflector_mspct" objects.
• valleys(solute_mspct): Method for "solute_mspct" objects.
• valleys(cps_mspct): Method for "cps_mspct" objects.
• valleys(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)

Arguments

x numeric vector (sorted in increasing order).
y numeric vector.
h a numeric vector giving the wavelengths at which the y values should be inserted by interpolation, no interpolation is indicated by an empty numeric vector (numeric(0)).

Value

A numeric vector with the numeric values of y, but longer. Unless the hinge values were already present in y, each inserted hinge, expands the vector by two values.

See Also

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

v_replace_hinges | Overwrite spectral data values at existing wavelength values.

Description

Overwriting spectral data with interpolated values at wavelengths values containing bad data is needed when cleaning spectral data. This function differs from insert_hinges() in that it returns a vector of y values instead of a tibble.
water.spct

Usage

v_replace_hinges(x, y, h)

Arguments

x numeric vector (sorted in increasing order).
y numeric vector.
h a numeric vector giving the wavelengths at which the y values should be replaced by interpolation, no interpolation is indicated by an empty numeric vector (numeric(0)).

Value

A numeric vector with the numeric values of y with values at the hinges replaced by interpolation of neighbours.

See Also

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

water.spct  Molar spectral attenuation coefficient of water

Description

A dataset containing the wavelengths at a 2 nm interval and the corresponding attenuation coefficients.

Usage

water.spct

Format

A solute_spct object with 251 rows and 2 variables

Details

- w.length (nm), range 300 to 800 nm.
- K.mole (cm-1/M)
Author(s)
Buiteveld et al. (1994) (original data)

References
https://omlc.org/spectra/water/

See Also

Examples
head(water.spct)
summary(water.spct)
solute_properties(water.spct)
cat(comment(water.spct))

---

**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)
```
water_mvc2vp(water.mvc, temperature)

water_vp2RH(
  water.vp,
  temperature,
  over.ice = FALSE,
  method = "tetens",
  pc = TRUE,
  check.range = TRUE
)

water_RH2vp(
  relative.humidity,
  temperature,
  over.ice = FALSE,
  method = "tetens",
  pc = TRUE,
  check.range = TRUE
)

water_vp_sat_slope(
  temperature,
  over.ice = FALSE,
  method = "tetens",
  check.range = TRUE,
  temperature.step = 0.1
)

psychrometric_constant(atmospheric.pressure = 101325)

Arguments

- **temperature**: numeric vector of air temperatures (C).
- **over.ice**: logical vector 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 ($gm^{-3}$).
- **pc**: logical flag for result returned as percent or not.
- **relative.humidity**: numeric Relative humidity as fraction of 1.
- **temperature.step**: numeric Delta or step used to estimate the slope as a finite difference (C).
- **atmospheric.pressure**: numeric Atmospheric pressure (Pa).
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.

Method "tetens" implements Tetens’ (1930) equation for the cases of equilibrium with a water and an ice surface. Method "magnus" implements the modified Magnus equations of Alduchov and Eskridge (1996, eqs. 21 and 23). Method "wexler" implements the equations proposed by Wexler (1976, 1977), and their inverse according to Hardy (1998). Method "goff.gratch" implements the equations of Groff and Gratch (1946) with the minor updates of Groff (1956).

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.

Value

A numeric vector of partial pressures in pascal (Pa) 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 (gm$^{-3}$) for `water_vp2mvc()`. `water_vp_sat()` and `psychrometric_constant()` both return numeric vectors of pressure per degree of temperature (PaC$^{-1}$)

Note

The inverse of the Groff Gratch equation has yet to be implemented.

References


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

water_dp(water.vp = 1000) # Pa -> C
water_dp(water.vp = 1000, method = "magnus") # Pa -> C
water_dp(water.vp = 1000, method = "wexler") # Pa -> C
water_dp(water.vp = 500, over.ice = TRUE) # Pa -> C
water_dp(water.vp = 500, method = "wexler", over.ice = TRUE) # Pa -> C

water_fp(water.vp = 300) # Pa -> C
water_dp(water.vp = 300, over.ice = TRUE) # Pa -> C

water_vp2RH(water.vp = 1500, temperature = 20) # Pa, C -> RH %
water_vp2RH(water.vp = 1500, temperature = c(20, 30)) # Pa, C -> RH %
water_vp2RH(water.vp = c(600, 1500), temperature = 20) # Pa, C -> RH %
water_vp2mvc(water.vp = 1000, temperature = 20) # Pa -> g m^-3
```
waveband

**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,
  SWF.norm = NULL,
  hinges = NULL,
  wb.name = NULL,
  wb.label = wb.name
)
```
waveband

Arguments

- `x`: any R object on which applying the method `range()` yields an vector of two numeric values, describing a range of wavelengths \([nm]\).
- `weight`: a character string "SWF" or "BSWF", use NULL (the default) to indicate no weighting used when calculating irradiance.
- `SWF.e.fun`, `SWF.q.fun`: a functions giving multipliers for a spectral weighting function (energy and quantum, respectively) as a function of wavelength \([nm]\).
- `norm`: a single numeric value indicating the wavelength \([nm]\) at which the SWF should be normalized to 1.0; NULL is interpreted as no normalization.
- `SWF.norm`: a numeric value giving the native normalization wavelength \([nm]\) used by `SWF.e.fun` and `SWF.q.fun`.
- `hinges`: a numeric vector giving the wavelengths at which values in `s.irrad` should be inserted by interpolation before integration is attempted. No interpolation is indicated by an empty vector (numeric(0)), while interpolation at both boundaries of the band is indicated by NULL.
- `wb.name`: character string giving the name for the waveband defined, default is NULL for an automatically generated name.
- `wb.label`: character string giving the label of the waveband to be used for labelling computed summaries or plots, default is `wb.name`.
- `w.low`, `w.high`: numeric value, wavelengths at the short end and long ends of the wavelength 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 irradiances in \([W m^{-2} nm^{-1}]\) or \([mol s^{-1} sm^{-2} nm^{-1}]\) as indicated by the argument passed to unit.in.
- **w.band.num**, **w.band.denom** waveband objects used to compute the numerator and denominator of the ratio.
- **unit.out.num**, **unit.out.denom** character Base of expression used to compute the numerator and denominator of the ratio. Allowed values "energy", and "photon", or its alias "quantum".
- **unit.in** character Allowed values "energy", and "photon", or its alias "quantum".
- **check.spectrum** logical Flag indicating whether to sanity check input data, default is TRUE.
- **use.cached.mult** logical Flag indicating whether multiplier values should be cached between calls.
- **use.hinges** logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value
a single numeric value giving the ratio
Note

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`. From version 0.9.19 onwards use of this default does not trigger a warning, but instead is used silently.

Examples

```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.hi` give the boundaries of the wavebands, `wl.color` the color definition corresponding to the wavelength at the center of the waveband and `wb.color` the color of the waveband as a whole (assuming a flat energy irradiance spectrum). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from `generic_spct`.

Note

Function `fast_wb2rect_spct()` differs from `wb2rect_spct()` in that it computes colors for narrow wavebands based on the midpoint wavelength and uses vectorization when possible. It always returns color definitions with short names, which are also used as waveband names for narrow wavebands and merged wavebands. The purpose of merging of rectangles is to speed up rendering and to reduce the size of vector graphics output. This function should be used with care as the color definitions returned are only approximate and original waveband names can be lost.

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `untag()`, `wb2spct()`, `wb2tagged_spct()`

---

**wb2spct**

Create spectrum from wavebands

---

**Description**

Create a `generic.spectrum` 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

Value
A generic.spectrum object, with columns w.length set to the union of all boundaries and hinges defined in the waveband(s). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from generic_spct.

See Also
Other tagging and related functions: is_tagged(), tag(), untag(), wb2rect_spct(), wb2tagged_spct()

wb2tagged_spct Create tagged spectrum from wavebands

Description
Create a tagged generic_spct object with wavelengths from the range of wavebands in a list, and names of the same bands as factor levels, and corresponding color definitions. The spectrum is not suitable for plotting labels, symbols, rectangles or similar, as the midpoint of each waveband is not added to the spectrum.

Usage
wb2tagged_spct(
  w.band,
  use.hinges = TRUE,
  short.names = TRUE,
  chroma.type = "CMF",
  ...
)

Arguments
  w.band waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are tagged and the wavelengths returned in variable w.length.
  use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
  short.names logical Flag indicating whether to use short or long names for wavebands.
  chroma.type character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system. Ignored (possibly used by derived methods).
  ...

Value
A spectrum as returned by wb2spct but additionally tagged using function tag

See Also
Other tagging and related functions: is_tagged(), tag(), untag(), wb2rect_spct(), wb2spct()
### wb_trim_as_default

**Set computation options**

**Description**

Set computation related options easily.

**Usage**

```r
ewb_trim\_as\_default(flag = TRUE)
\nuse\_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**

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`, `opaque.spct`, `phenylalanine.spct`, `photodiode.spct`, `polyester.spct`, `sun.daily.data`, `sun.daily.spct`, `sun.data`, `sun.spct`, `water.spct`, `white led cps.spct`, `white led raw.spct`, `white led source.spct`, `yellow gel.spct`
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

whiteLed.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

  white_led.source_spct

wls_at_target <- function(x, target = NULL, interpolate = FALSE, idfactor = length(target) > 1, na.rm = FALSE, ...) {
  ...
}

## Default S3 method:
wls_at_target(
  x,
  target = NULL, interpolate = FALSE, idfactor = length(target) > 1, na.rm = FALSE,
)

## S3 method for class 'data.frame'
wls_at_target(
  x,
  target = "0.5max", interpolate = FALSE,
)
wls_at_target

idfactor = length(target) > 1,
na.rm = FALSE,
x.var.name = NULL,
y.var.name = NULL,
...)

## S3 method for class 'generic_spct'
wls_at_target(
  x,
target = "0.5max",
interpolate = FALSE,
idfactor = length(target) > 1,
na.rm = FALSE,
col.name = NULL,
y.var.name = col.name,
...)

## S3 method for class 'source_spct'
wls_at_target(
  x,
target = "0.5max",
interpolate = FALSE,
idfactor = length(target) > 1,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...)

## S3 method for class 'response_spct'
wls_at_target(
  x,
target = "0.5max",
interpolate = FALSE,
idfactor = length(target) > 1,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...)

## S3 method for class 'filter_spct'
wls_at_target(
  x,
target = "0.5max",
interpolate = FALSE,
idfactor = length(target) > 1,
na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
...

## S3 method for class 'reflector_spct'

wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'solute_spct'

wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'

wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

## S3 method for class 'generic_mspct'

wls_at_target(
  x,
  target = "0.5max",
  interpolate = FALSE,
  idfactor = length(target) > 1,
  na.rm = FALSE,
  ...
)

Arguments

x data.frame or spectrum object.
target numeric or character vector. A numeric value indicates the spectral quantity value for which wavelengths are to be searched. A character string representing a number is converted to numeric. A character value representing a number followed by a function name, will be also accepted and decoded, such that "0.1max" is interpreted as targeting one tenth of the maximum value in the column. The character strings "half.maximum" and "HM" are synonyms for "0.5max" while "half.range" and "HR" are synonyms for "0.5range".

interpolate logical Indicating whether the nearest wavelength value in \textit{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 \texttt{idfactor = TRUE} then the column is auto named \texttt{target.idx}. Alternatively the column name can be directly passed as argument to \texttt{idfactor} as a character string.

na.rm logical indicating whether NA values should be stripped before searching for the target.

... currently ignored.

\texttt{x.var.name, y.var.name, col.name}

character The name of the columns in which to search for the target value. Use of \texttt{col.name} is deprecated, and is a synonym for \texttt{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 \texttt{.export} and \texttt{.packages} arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A data.frame, a spectrum object or a collection of spectra object of the same class as \texttt{x} with fewer rows, possibly even no rows. If \texttt{FALSE} is passed to \texttt{interpolate} a subset of \texttt{x} is returned, otherwise a new object of the same class containing interpolated wavelengths for the target value is returned. As 'target' accepts a vector or list as argument, a factor can be added to the output with the corresponding target value.

Methods (by class)

- \texttt{wls_at_target(default)}: Default returning always an empty object of the same class as \texttt{x}.
- \texttt{wls_at_target(data.frame)}: Method for "data.frame" objects.
- \texttt{wls_at_target(generic_spct)}: Method for "generic_spct" objects.
- \texttt{wls_at_target(source_spct)}: Method for "source_spct" objects.
- \texttt{wls_at_target(response_spct)}: Method for "response_spct" objects.
- \texttt{wls_at_target(filter_spct)}: Method for "filter_spct" objects.
- \texttt{wls_at_target(reflector_spct)}: Method for "reflector_spct" objects.
• `wls_at_target(solute_spct)`: Method for "solute_spct" objects.
• `wls_at_target(cps_spct)`: Method for "cps_spct" objects.
• `wls_at_target(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")
```

---

\[ wl\_max \]

**Wavelength maximum**

**Description**

A method specialization that returns the wavelength maximum [\text{nm}] 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
- **...**: numeric, `waveband` or `generic_spct` arguments.
- **idx**: character Name of the column with the names of the members of the collection of spectra.

Value

- a length-one vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- `max(generic_spct)`:
- `max(generic_mspct)`:

Examples

```r
max(sun.spct)
wl_max(sun.spct)
```

### Description

A method that returns the wavelength [nm] (or value) at the center of the wavelength range of objects of classes `waveband` or of class `generic_spct` or derived (or the midpoint from a numeric vector).

### Usage

```r
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 \( \frac{\text{max}(x) - \text{min}(x)}{2} \). In the case of spectral objects a wavelength [nm]. For any other R object, according to available definitions of \text{min} and \text{max}.

### Methods (by class)

- **midpoint(default)**: Default method for generic function
- **midpoint(numeric)**: Default method for generic function
- **midpoint(waveband)**: Wavelength at center of a "waveband".
- **midpoint(generic_spct)**: Method for "generic_spct".
- **midpoint(generic_mspct)**: Method for "generic_mspct" objects.

### See Also

Other wavelength summaries: \text{wl_min()}, \text{wl_range()}, \text{wl_stepsize()}

### Examples

```r
midpoint(10:20)
midpoint(sun.spct)
w1_midpoint(sun.spct)
midpoint(sun.spct)
```
Description

A method specialization that returns the wavelength minimum \([nm]\) from objects of classes \texttt{waveband}\ or of class \texttt{generic\_spct} or derived.

Usage

\begin{verbatim}
wl_min(x, na.rm = FALSE)
## S3 method for class \texttt{waveband}
min(..., na.rm = FALSE)
## S3 method for class \texttt{generic\_spct}
min(..., na.rm = FALSE)
## S3 method for class \texttt{generic\_mspct}
min(..., na.rm = FALSE, idx = "spct.idx")
\end{verbatim}

Arguments

- \texttt{x} \hspace{1cm} \texttt{generic\_spct, generic\_mspct or waveband object.}
- \texttt{na.rm} \hspace{1cm} ignored
- \texttt{...} \hspace{1cm} not used in current version
- \texttt{idx} \hspace{1cm} character Name of the column with the names of the members of the collection of spectra.

Value

a length-one vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- \texttt{min(generic\_spct)}:
- \texttt{min(generic\_mspct)}:

See Also

Other wavelength summaries: \texttt{wl\_midpoint}, \texttt{wl\_range}, \texttt{wl\_stepsize}()

Examples

\begin{verbatim}
min(sun.spct)
wl_min(sun.spct)
\end{verbatim}
Description

A method specialization that returns the wavelength range [nm] from objects of classes waveband or of class generic_spct or derived.

Usage

\[
\text{wl\_range}(x, \text{na.rm} = \text{FALSE})
\]

## S3 method for class 'waveband'
range(..., na.rm = FALSE)

## S3 method for class 'generic_spct'
range(..., na.rm = FALSE)

## S3 method for class 'generic_mspct'
range(..., na.rm = FALSE, idx = "spct.idx")

Arguments

- **x**: generic_spct, generic_mspct or waveband object.
- **na.rm**: ignored
- **...**: a single R object
- **idx**: character Name of the column with the names of the members of the collection of spectra.

Value

a length-two vector for individual objects or numeric vectors or a data frame for collections of spectra.

Methods (by class)

- range(generic_spct):
- range(generic_mspct):

See Also

Other wavelength summaries: \text{wl\_midpoint()}, \text{wl\_min()}, \text{wl\_stepsize()}

Examples

\begin{verbatim}
range(sun.spct)
wl_range(sun.spct)
range(sun.spct)
\end{verbatim}

\begin{tabular}{lc}
\hline
wl_stepsize & \emph{Stepsize} \\
\hline
\end{tabular}

Description

Method returning the range of step sizes in an object; i.e., the Range of differences between successive sorted values. In particular the wavelength step sizes [\textit{nm}] of objects of class \texttt{generic_spct} or derived (or the step sizes of values in a numeric vector).

Usage

\begin{verbatim}
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")
\end{verbatim}

Arguments

\begin{verbatim}
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.
\end{verbatim}

Value

A numeric vector of length 2 with min and maximum stepsize values.
Methods (by class)

- `stepsize(default)`: Default function usable on numeric vectors.
- `stepsize(numeric)`: Method for numeric vectors.
- `stepsize(generic_spct)`: Method for "generic_spct" objects.
- `stepsize(generic_mspct)`: Method for "generic_mspct" objects.

See Also

Other wavelength summaries: `wl_midpoint()`, `wl_min()`, `wl_range()`

Examples

```r
stepsize(sun.spct)
w1_stepsize(sun.spct)
stepsize(sun.spct)
```

---

**w_length2rgb**

_Wavelength to rgb color conversion_

Description

Calculates rgb values from spectra based on human color matching functions

Usage

```r
w_length2rgb(w.length, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)
```

Arguments

- `w.length`: numeric Vector of wavelengths [nm].
- `sens`: chroma_spct Used as chromaticity definition.
- `color.name`: character Used for naming the rgb color definition.

Value

A vector of colors defined using `rgb()`. The numeric values of the RGB components can be obtained using function `col2rgb()`.

See Also

Other color functions: `rgb_spct()`, `w_length_range2rgb()`
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  

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**

```
## 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

* **BSWF attribute functions**
  - getBSWFUsed, 164
  - setBSWFUsed, 291

* **Coercion methods for collections of spectra**
  - as.calibration_mspct, 24
  - as.chroma_mspct, 27
  - as cps_mspct, 28
  - as.filter_mspct, 31
  - as.generic_mspct, 34
  - as.object_mspct, 38
  - as.raw_mspct, 40
  - as.reflector_mspct, 42
  - as.response_mspct, 45
  - as.solute_mspct, 49
  - as.source_mspct, 52
  - split2mspct, 333
  - subset2mspct, 343

* **Evapotranspiration and energy balance related functions.**
  - ET_ref, 123
  - net_irradiance, 229

* **K attribute functions**
  - getKType, 170
  - setKType, 301

* **Local solar time functions**
  - as.solar_date, 48
  - is.solar_time, 204
  - print.solar_time, 256
  - solar_time, 318

* **Rfr attribute functions**
  - getRfrType, 174
  - setRfrType, 304

* **Spectral data examples**
  - A.illuminant_spct, 11
  - black_body_spct, 60
  - ccd_spct, 63
  - clear_spct, 81
  - clear_body_spct, 82
  - D65.illuminant_spct, 98
  - green_leaf_spct, 186
  - Ler_leaf_spct, 216
  - Ler_leaf_rflt_spct, 217
  - Ler_leaf_trns_spct, 218
  - Ler_leaf_trns_i_spct, 219
  - opaque_spct, 239
  - phenylalanine_spct, 248
  - photodiode_spct, 249
  - polyester_spct, 255
  - sun.daily_data, 348
  - sun.daily_spct, 349
  - sun_data, 350
  - sun_spct, 351
  - water_spct, 409
  - white_body_spct, 420
  - white_ledcps_spct, 421
  - white_ledraw_spct, 421
  - white_ledsource_spct, 422
  - yellow_gel_spct, 435

* **Tfr attribute functions**
  - getTfrType, 177
  - setTfrType, 308

* **Time of day functions**
  - as_tod, 58
  - format.tod_time, 151
  - print.tod_time, 258

* **Visual response data examples**
  - beesxyzCMF_spct, 59
  - ciev10_spct, 70
  - ciev2_spct, 71
  - ciexyzCC10_spct, 72
  - ciexyzCC2_spct, 73
  - ciexyzCMF10_spct, 74
  - ciexyzCMF2_spct, 75
  - cone_fundamentals10_spct, 89

* **astronomy related functions**
  - day_night, 99
  - format.solar_time, 151
sun_angles, 352
* auxiliary functions
  normalize_range_arg, 238
* collections of spectra classes family
  generic_mspct, 163
* color functions
  rgb_spect, 286
  w_length2rgb, 433
  w_length_range2rgb, 434
* constructors of spectral objects
  as.calibration_spct, 26
  as.chroma_spct, 28
  as.cps_spct, 30
  as.filter_spct, 33
  as.generic_spct, 36
  as.object_spct, 39
  as.raw_spct, 42
  as.reflector_spct, 44
  as.response_spct, 47
  as.solute_spct, 51
  as.source_spct, 54
  source_spct, 319
* conversion of collections of spectra
  join_mspct, 214
* data validity check functions
  check_spct, 65
  check_spectrum, 68
  check_w.length, 69
  enable_check_spct, 117
* datasets
  A.illuminant.spct, 11
  beesxyzCMF.spct, 59
  black_body.spct, 60
  ccd.spct, 63
  ciev10.spct, 70
  ciev2.spct, 71
  ciexyzCC10.spct, 72
  ciexyzCC2.spct, 73
  ciexyzCMF10.spct, 74
  ciexyzCMF2.spct, 75
  clear.spct, 81
  clear_body.spct, 82
  cone_fundamentals10.spct, 89
  D2.UV586, 96
  D2.UV653, 96
  D2.UV654, 97
  D65.illuminant.spct, 98
  FEL.BN.9101.165, 141
  green_leaf.spct, 186
  Ler_leaf.spct, 216
  Ler_leaf.rflt.spct, 217
  Ler_leaf.trns.spct, 218
  Ler_leaf.trns_i.spct, 219
  opaque.spct, 239
  phenylalanine.spct, 248
  photodiode.spct, 249
  polyester.spct, 255
  r4p_pkgs, 275
  sun.daily.data, 348
  sun.daily.spct, 349
  sun.data, 350
  sun.spct, 351
  water_spct, 409
  white_body.spct, 420
  white_led.cps_spct, 421
  white_led.raw_spct, 421
  white_led.source_spct, 422
  yellow_gel.spct, 435
* despike and valleys functions
  despike, 103
* experimental utility functions
  collect2mspct, 84
  drop_user_cols, 113
  thin_wl, 378
  uncollect2spct, 395
* idfactor attribute functions
  getIdFactor, 168
  setIdFactor, 298
* internal.
  v_insert_hinges, 408
  v_replace_hinges, 408
* interpolate functions
  interpolate_wl, 194
* irradiance functions
  e_fluence, 129
  e_irrad, 132
  fluence, 148
  irrad, 196
  q_fluence, 264
  q_irrad, 266
* low-level functions operating on numeric vectors.
  as_energy, 56
  as_quantum_mol, 57
  calc_multipliers, 61
  div_spectra, 112
INDEX

energy_irradiance, 119
energy_ratio, 120
insert_hinges, 188
integrate_xy, 191
interpolate_spectrum, 193
irradiance, 199
operate_spectra, 240
photon_irradiance, 251
photon_ratio, 253
photons_energy_ratio, 250
produce_spectra, 259
s_e_irradi2rgb, 355
split_energy_irradiance, 337
split_photon_irradiance, 340
subt_spectra, 344
sum_spectra, 347
v_insert_hinges, 408
v_replace_hinges, 408

* math operators and functions
  ^_.generic_spct, 435
  convolve_each, 93
  div_.generic_spct, 111
  log, 220
  MathFun, 221
  minus_.generic_spct, 224
  mod_.generic_spct, 224
  plus_.generic_spct, 254
  round, 289
  sign, 314
  slash_.generic_spct, 315
  times_.generic_spct, 381

* measurement metadata functions
  add_attr2tb, 19
  get_attributes, 183
  getInstrProperties, 165
  getHowMeasured, 166
  getInstrDesc, 168
  getInstrSettings, 169
  getSoluteProperties, 175
  getWhatMeasured, 179
  getWhenMeasured, 180
  getWhereMeasured, 181
  isValidInstrDesc, 206
  isValidInstrSettings, 207
  select_spct_attributes, 290
  setFilterProperties, 292
  setHowMeasured, 297
  setInstrDesc, 299
  setInstrSettings, 300
  setSoluteProperties, 306
  setWhatMeasured, 310
  setWhenMeasured, 311
  setWhereMeasured, 312
  spct_attr2tb, 325
  spct_metadata, 326
  trimInstrDesc, 385
  trimInstrSettings, 386

* multiple.wl attribute functions
  getMultipleWL, 171
  setMultipleWL, 301

* peaks and valleys functions
  find_peaks, 142
  find_spikes, 143
  get_peaks, 184
  peaks, 241
  replace_bad_pixs, 281
  spikes, 328
  valleys, 400
  wls_at_target, 423

* photon and energy ratio functions
  e_ratio, 135
  eq_ratio, 121
  q_ratio, 270
  qe_ratio, 261

* quantity conversion functions
  A2T, 12
  Afr2T, 21
  any2T, 23
  as_quantum, 56
  e2q, 115
  e2qmol_multipliers, 116
  e2quantum_multipliers, 117
  q2e, 260
  T2A, 372
  T2Afr, 374

* query units functions
  is_absorbance_based, 208
  is_mole_based, 210
  is_photon_based, 211

* rescaling functions
  fscale, 152
  fshift, 158
  getNormalized, 172
  getScaled, 174
  is_normalized, 211
is_scaled, 212
normalize, 231
setNormalized, 302
setScaled, 305

* response functions
  e_response, 138
  q_response, 272
  response, 283

* response type attribute functions
  setResponseType, 303

* response.type attribute functions
  getResponseType, 173

* set and unset 'multi spectral' class functions
  rmDerivedMspct, 287
  shared_member_class, 314

* set and unset spectral class functions
  rmDerivedSpct, 288
  setGenericSpct, 293

* split a spectrum into regions functions
  split_irradiance, 338

* tagging and related functions
  is_tagged, 213
  tag, 370
  untag, 396
  wb2rect_spct, 417
  wb2spct, 418
  wb2tagged_spct, 419

* time attribute functions
  checkTimeUnit, 64
  convertTfrType, 90
  convertThickness, 91
  convertTimeUnit, 92
  getTimeUnit, 178
  setTimeUnit, 309

* trim functions
  clip_wl, 83
  trim_spct, 387
  trim_waveband, 390
  trim_wl, 392

* upgrade from earlier versions
  is_old_spct, 204
  upgrade_spct, 397
  upgrade_spectra, 398

* waveband attributes
  is_effective, 209
  labels, 216
  normalization, 230

* waveband constructors
  split_bands, 335
  waveband, 414

* wavelength summaries
  wl_midpoint, 428
  wl_min, 430
  wl_max, 431
  wl_stepsizing, 432

*.generic_spct (times-.generic_spct), 381
+.generic_spct (plus-.generic_spct), 254
-.generic_spct (minus-.generic_spct), 224
/.generic_spct (slash-.generic_spct), 315
[.chroma_spct (Extract), 126
[.cps_spct (Extract), 126
[.filter_spct (Extract), 126
[.generic_mspct (Extract_mspct), 128
[.generic_spct (Extract), 126
[.object_spct (Extract), 126
[.raw_spct (Extract), 126
[.reflector_spct (Extract), 126
[.response_spct (Extract), 126
[.solute_spct (Extract), 126
[.source_spct (Extract), 126
[<-.generic_mspct (Extract_mspct), 128
[<-.generic_spct (Extract), 126
[[<-.generic_mspct (Extract_mspct), 128
$<-.generic_mspct (Extract_mspct), 128
$<-.generic_spct (Extract), 126
%%.generic_spct (div-.generic_spct), 111
%.generic_spct (mod-.generic_spct), 224
^generic_spct, 93, 112, 221, 224, 254, 290, 314, 315, 381, 435
A.illuminant.spct, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
A2T, 12, 23, 24, 57, 116, 117, 261, 374, 375
A_as_default (energy_as_default), 118
abs.generic_spct (MathFun), 221
absorbance, 13, 237
absorptance, 16
acos.generic_spct (Trig), 384
add_attr2tb, 14, 17, 19, 122, 131, 134, 137, 139, 150, 166, 167, 169, 176, 180–182, 184, 198, 207, 262, 265,
chroma_spct (source_spct), 319
ciev10.spct, 60, 70, 71–75, 90
criev2.spct, 60, 70, 71, 72–75, 90
criexyzCC10.spct, 60, 70, 71, 72–75, 90
ciexyzCC2.spct, 60, 70–72, 73, 74, 75, 90
criexyzCMF10.spct, 60, 70–73, 74, 75, 90
ciexyzCMF2.spct, 60, 70–74, 75, 90
class_spct, 76
clean, 76
clear.spct, 11, 60, 64, 81, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
clear_body.spct, 11, 60, 64, 81, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
clip_wl, 83, 388, 391, 394
col2rgb, 355, 434
collect2mspct, 84, 115, 380, 396
color (color_of), 85
color_of, 85
colour_of (color_of), 85
comment2t2b (add_attr2t2b), 19
cmpare_spct, 87
cone_fundamentals10.mspct
  (cone_fundamentals10.spct), 89
cone_fundamentals10.spct, 60, 70–75, 89
cor, 372
cos.generic_spct (Trig), 384
cps2irrad, 95
cps2Rfr (cps2irrad), 95
cps2Tfr (cps2irrad), 95
cps_mspct (generic_mspct), 163
cps_spct (source_spct), 319

d2.UV586, 96
d2.UV653, 96
d2.UV654, 97
d2_spectrum, 97
d65.illuminant.spct, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
day_length (day_night), 99
day_night, 99, 151, 354
day_night_fast (day_night), 99
defunct, 102
despike, 103
diffraction_double_slit
  (diffraction_single_slit), 110
diffraction_single_slit, 110
dim.generic_mspct, 111
dim<-.generic_mspct
  (dim.generic_mspct), 111
disable_check_spct (enable_check_spct), 117
distance_to_sun (sun_angles), 352
div-.generic_spcct, 111
drop_user_cols, 85, 113, 380, 396
e2q, 13, 23, 24, 57, 115, 116, 117, 261, 374, 375
e2qmol_multipliers, 13, 23, 24, 57, 116, 117, 261, 374, 375
e2quantum_multipliers, 113, 23, 24, 57, 116, 117, 261, 374, 375
e fluorescence, 129, 134, 150, 199, 266, 269
e_irrad, 131, 150, 199, 266, 269
e_ratio, 123, 135, 263, 272
e_response, 138, 275, 285
enable_check_spct, 68, 69, 117, 296
energy_as_default, 118
eq Ratio, 121, 138, 263, 272
ET_ref, 123, 230
ET_ref_day (ET_ref), 123
exp.generic_spct (log), 220
expans (spread), 341
extend2extremes (trim_spct), 387
Extract, 126, 127, 128
Extract_mspct, 128
Extremes, 365
interpolate_mspct (interpolate_spct), 192
interpolate_spct, 192
interpolate_wl, 194
irrad, 131, 134, 150, 196, 266, 269
irrad_extraterrestrial, 200
is.any_mspct (is.generic_mspct), 201
is.any_spct (is.generic_spct), 203
is.any_summary_spct
  (is.summary_generic_spct), 205
is.calibration_mspct
  (is.generic_mspct), 201
is.calibration_spct (is.generic_spct), 203
is.chroma_mspct (is.generic_mspct), 201
is.chroma_spct (is.generic_spct), 203
is.cps_mspct (is.generic_mspct), 201
is.cps_spct (is.generic_spct), 203
is.filter_mspct (is.generic_mspct), 201
is.filter_spct (is.generic_spct), 203
is.filtered_mspct (is.generic_mspct), 201
is.filtered_spct (is.generic_spct), 203
is.generic_mspct, 201
is.generic_spct, 203
is.object_mspct (is.generic_mspct), 201
is.object_spct (is.generic_spct), 203
is.old_spct, 204, 397, 398
is.raw_mspct (is.generic_mspct), 201
is.raw_spct (is.generic_spct), 203
is.reflector_mspct (is.generic_mspct), 201
is.reflector_spct (is.generic_spct), 203
is.response_mspct (is.generic_mspct), 201
is.response_spct (is.generic_spct), 203
is.solar_date (is.solar_time), 204
is.solar_time, 48, 204, 257, 319
is.solute_mspct (is.generic_mspct), 201
is.solute_spct (is.generic_spct), 203
is.source_mspct (is.generic_mspct), 201
is.source_spct (is.generic_spct), 203
is.summary_chroma_spct
  (is.summary_generic_spct), 205
is.summary_cps_spct
  (is.summary_generic_spct), 205
is.summary_filter_spct
  (is.summary_generic_spct), 205
is.summary_generic_spct, 205
is.summary_object_spct
  (is.summary_generic_spct), 205
is.summary_raw_spct
  (is.summary_generic_spct), 205
is.summary_reflector_spct
  (is.summary_generic_spct), 205
is.summary_response_spct
  (is.summary_generic_spct), 205
is.summary_solute_spct
  (is.summary_generic_spct), 205
is.summary_source_spct
  (is.summary_generic_spct), 205
is.waveband, 206
is_absorbance_based, 208, 210, 212
is_absorptance_based
  (is_absorbance_based), 208
is_effective, 209, 216, 231
is_energy_based (is_photon_based), 211
is_mass_based (is_mole_based), 210
is_mole_based, 208, 210, 212
is_normalised (is_normalized), 211
is_normalized, 158, 162, 172, 175, 211, 213,
normalized2tb (add_attr2tb), 19
normalized_diff_ind, 237

object_mspct (generic_mspct), 163
object_spct (source_spct), 319
opaque_spct, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435

paste, 152
peaks, 143, 144, 185, 241, 282, 333, 407, 427
photobiology (photobiology-package), 9
photodiode_spct, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 249, 255, 349–352, 410, 420–423, 435
photon_as_default (energy_as_default), 118
plus_.generic_spct, 254
polyester_spct, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
print, 255
print.solar_date (print.solar_time), 256
print.solar_time, 48, 205, 256, 319
print.summary.generic_spct, 257
print.tod_time, 58, 151, 258
print.waveband, 258
prod, 364
psychrometric_constant (water_vp_sat), 410
q2e, 13, 23, 24, 57, 116, 117, 260, 374, 375
q_fluence, 131, 134, 150, 199, 264, 269
q_irrad, 131, 134, 150, 199, 266, 266
q_ratio, 123, 138, 263, 270
q_response, 140, 272, 285
qe_ratio, 123, 138, 261, 272
quantum_as_default (energy_as_default), 118
r4p_pkgs, 275
range, 152
range (wl_range), 431
raw_mspct (generic_mspct), 163
raw_spct (source_spct), 319
rbindspct, 276
reflectance, 237, 277
reflector_mspct (generic_mspct), 163
reflector_spct (source_spct), 319
relative_AM, 280
replace_bad_pixs, 109, 143, 144, 185, 248, 281, 333, 407, 427
response, 140, 237, 275, 283
response_mspct (generic_mspct), 163
response_spct (source_spct), 319
Rfr_from_n, 285
Rfr_p_from_n (Rfr_from_n), 285
Rfr_s_from_n (Rfr_from_n), 285
Rfr_type2tb (add_attr2tb), 19
rgb, 355
rgb_spct, 286, 433, 434
rmDerivedMspct, 287, 314
rmDerivedSpct, 288, 297
round, 93, 112, 221, 224, 254, 289, 314, 315, 381, 436
s_mean, 356
s_mean_se, 358
s_mean_se_band, 359
INDEX

s_median, 361
s_prod, 362
s_range, 364
s_sd, 366
s_se, 367
s_sum, 369
s_var, 371
scaled2tb (add_attr2tb), 19
sd, 367
select_spct_attributes, 21, 166, 167, 169,
176, 180–182, 184, 207, 290, 293,
298, 300, 307, 310, 312, 313, 325,
327, 385, 386
set_check_spct (enable_check_spct), 117
setAfrType (defunc), 102
setBSWFUsed, 165, 291
setCalibrationSpct (setGenericSpct), 293
setChromaSpct (setGenericSpct), 293
setCpsSpct (setGenericSpct), 293
setFilterProperties, 21, 166, 167, 169,
176, 180–182, 184, 207, 291, 293,
298, 300, 307, 310, 312, 313, 325,
327, 385, 386
setFilterSpct (setGenericSpct), 293
setGenericSpct, 26, 28, 31, 34, 37, 40, 42,
45, 48, 55, 288, 293
setHowMeasured, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 297,
300, 307, 310, 312, 313, 325, 327,
385, 386
setIdFactor, 168, 298
setInstrDesc, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 297,
299, 300, 307, 310, 312, 313, 325,
327, 385, 386
setInstrSettings, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 297,
300, 300, 307, 310, 312, 313, 325,
327, 385, 386
setKType, 170, 301
setMultipleWl, 171, 301
setNormalised (setNormalized), 302
setNormalized, 158, 162, 172, 175, 211, 213,
236, 302, 306
setObjectSpct (setGenericSpct), 293
setRawSpct (setGenericSpct), 293
setRefelectorSpct (setGenericSpct), 293
setResponseSpct (setGenericSpct), 293
setResponseType, 303
setRfrType, 174, 304
setScaled, 158, 162, 172, 175, 211, 213, 236,
303, 305
setSoluteProperties, 21, 166, 167, 169,
176, 180–182, 184, 207, 291, 293,
298, 300, 306, 310, 312, 313, 325,
327, 385, 386
setSoluteSpct, 52
setSoluteSpct (setGenericSpct), 293
setSourceSpct (setGenericSpct), 293
setTfrType, 178, 308
setTimeUnit, 65, 91–93, 178, 309
setWhatMeasured, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 298,
300, 307, 310, 312, 313, 325, 327,
385, 386
setWhenMeasured, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 298,
300, 307, 310, 311, 313, 325, 327,
385, 386
setWhereMeasured, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 298,
300, 307, 310, 312, 313, 325, 327,
385, 386
shared_member_class, 288, 314
sign, 93, 112, 221, 224, 254, 290, 314, 315,
381, 436
signif.generic_spct (round), 289
sin.generic_spct (Trig), 384
slash generic_spct, 315
smooth_spct, 315
solar_time, 48, 58, 205, 257, 318
solute_mspct (generic_mspct), 163
solute_properties
 (getSoluteProperties), 175
solute_properties
 (setSoluteProperties), 306
solute_spct (source_spct), 319
source_mspct (generic_mspct), 163
source_spct, 26, 28, 31, 34, 37, 40, 42, 45,
48, 52, 55, 319
spct_attr2tb, 21, 166, 167, 169, 176,
180–182, 184, 207, 291, 293, 298,
300, 307, 310, 312, 313, 325, 327,
385, 386
spct_attributes
 (select_spct_attributes), 290
spct\_classes, \texttt{76, 326}
spct\_metadata, \texttt{21, 166, 167, 169, 176}, \texttt{180–182, 184, 207, 291, 293, 298}, \texttt{300, 307, 310, 312, 313, 325, 326}, \texttt{385, 386}
spikes, \texttt{143, 144, 185, 248, 282, 328, 407, 427}
splinefun, \texttt{194}
split2calibration\_mspct (split2mspct), \texttt{333}
split2cps\_mspct (split2mspct), \texttt{333}
split2filter\_mspct (split2mspct), \texttt{333}
split2mspct, \texttt{26, 27, 30, 32, 36, 41, 44}, \texttt{47, 50, 54, 333, 344}
split2raw\_mspct (split2mspct), \texttt{333}
split2reflector\_mspct (split2mspct), \texttt{333}
split2response\_mspct (split2mspct), \texttt{333}
split2solute\_mspct (split2mspct), \texttt{333}
split2source\_mspct (split2mspct), \texttt{333}
split\_bands, \texttt{335, 415}
split\_energy\_irradiance, \texttt{56, 57, 62, 113}, \texttt{119, 121, 189, 191, 194, 200, 241}, \texttt{251, 252, 254, 260, 337, 341, 345}, \texttt{348, 356, 390, 408, 409}
split\_irradiance, \texttt{338}
spread, \texttt{341}
sqrt\_generic\_spect (MathFun), \texttt{221}
steps\_size (wl\_steps\_size), \texttt{432}
strict\_range\_as\_default
\hspace{1em} (verbose\_as\_default), \texttt{407}
\texttt{Summary}, \texttt{342}
subset, \texttt{128}
subset\_generic\_spect (Subset), \texttt{342}
subset2mspct, \texttt{26, 27, 30, 32, 36, 39, 41, 44}, \texttt{47, 50, 54, 335, 343}
sun\_angles, \texttt{101, 151, 201, 352}
sun\_angles\_fast (sun\_angles), \texttt{352}
sun\_azimuth (sun\_angles), \texttt{352}
sun\_elevation (sun\_angles), \texttt{352}
sun\_zenith\_angle (sun\_angles), \texttt{352}
sunrise\_time (day\_night), \texttt{99}
sunset\_time (day\_night), \texttt{99}
Sys\_timezone, \texttt{395}
T2A, \texttt{13, 23, 24, 57, 116, 117, 261, 372, 375}
T2Afr, \texttt{13, 23, 24, 57, 116, 117, 261, 374, 374}
T2T (defunct), \texttt{102}
tag, \texttt{88, 213, 376, 397, 418, 419}
tan\_generic\_spect (Trig), \texttt{384}
Tfr\_as\_default (energy\_as\_default), \texttt{118}
Tfr\_type2tb (add\_attr2tb), \texttt{19}
thin\_wl, \texttt{85, 115, 378, 396}
time\_unit2tb (add\_attr2tb), \texttt{19}
times\_\_\_generic\_spect, \texttt{381}
transmittance, \texttt{237, 381}
Trig, \texttt{384}
trim2overlap (trim\_spect), \texttt{387}
trim\_mspct (trim\_spect), \texttt{387}
trim\_spect, \texttt{84, 128, 387, 391, 394}
trim\_waveband, \texttt{84, 388, 390, 394}
trim\_wl, \texttt{84, 388, 391, 392}
trim\_InstrDesc, \texttt{21, 166, 167, 169, 176}, \texttt{180–182, 184, 207, 291, 293, 298}, \texttt{300, 307, 310, 312, 313, 325, 327, 385, 386}
trim\_InstrSettings, \texttt{21, 166, 167, 169, 176}, \texttt{180–182, 184, 207, 291, 293, 298}
INDEX

300, 307, 310, 312, 313, 325, 327, 385, 386
tnc2spct (round), 289
tz_time_diff, 394

uncollect2spct, 85, 115, 380, 395
unset_filter_qy_default
(energy_as_default), 118
unset_radiation_unit_default
(energy_as_default), 118
unset_userdefaults
(energy_as_default), 118
untag, 213, 377, 396, 418, 419
upgrade_spectra, 204, 397, 398
upgrade_spectra, 204, 397, 398
use_cached_mult_as_default
(wb_trim_as_default), 420
using_A (using_Tfr), 398
using_Afr (using_Tfr), 398
using_energy (using_Tfr), 398
using_photon (using_Tfr), 398
using_quantum (using_Tfr), 398
using_Tfr, 398

valleys, 143, 144, 185, 248, 282, 333, 400, 427
verbose_as_default, 407

w_length2rgb, 287, 433, 434
w_length_range2rgb, 287, 433, 434
water_spect, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 409, 420–423, 435
water_dp (water_vp_sat), 410
water_fp (water_vp_sat), 410
water_mvc2vp (water_vp_sat), 410
water_RH2vp (water_vp_sat), 410
water_vp2mvc (water_vp_sat), 410
water_vp2RH (water_vp_sat), 410
water_vp_sat, 410
water_vp_sat_slope (water_vp_sat), 410
waveband, 86, 231, 336, 414
waveband_ratio, 416
wb2rect_spectra, 213, 377, 397, 417, 419
wb2spectra, 213, 377, 397, 418, 419
wb2tagged_spectra, 213, 377, 397, 418, 419, 419
wb_trim_as_default, 420
what_measured (getWhatMeasured), 179
what_measured2tb (add_attr2tb), 19
what_measured<-- (setWhatMeasured), 310
when_measured (getWhenMeasured), 180
when_measured2tb (add_attr2tb), 19
when_measured<-- (setWhenMeasured), 311
where_measured (getWhereMeasured), 181
where_measured<-- (setWhereMeasured), 312
white_body_spectra, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435
white_led_cps_spectra, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420, 421, 422, 423, 435
white_led_raw_spectra, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420, 421, 423, 427, 435
white_led_source_spectra, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–422, 422, 435
wl_expanse (spread), 341
wl_max, 427
wl_midpoint, 428, 430, 431, 433
wl_min, 429, 430, 431, 433
wl_range, 429, 430, 431, 433
wl_stepsize, 429–431, 432
wls_at_target, 143, 144, 185, 248, 282, 333, 407, 423

yellow_gel_spectra, 11, 60, 64, 82, 98, 186, 217–220, 239, 249, 250, 255, 349–352, 410, 420–423, 435