

Package ‘sarp.snowprofile’

February 10, 2022

Version 1.2.1

Date 2022-02-07

Title Snow Profile Analysis for Snowpack and Avalanche Research

Description Analysis and plotting tools for snow profile data produced from manual snowpack observations and physical snowpack models. The functions in this package support snowpack and avalanche research by reading various formats of data (including CAAML, SMET, generic csv, and outputs from the snow cover model SNOWPACK), manipulate the data, and produce graphics such as stratigraphy and time series profiles. Package developed by the Simon Fraser University Avalanche Research Program <<http://www.avalancheresearch.ca>>.

Graphics apply visualization concepts from Horton, Nowak, and Haegeli (2020, <[doi:10.5194/nhess-20-1557-2020](https://doi.org/10.5194/nhess-20-1557-2020)>).

URL <http://www.avalancheresearch.ca>

License CC BY-SA 4.0

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Language en-CA

Imports data.table, methods, xml2,

Depends R (>= 2.10)

Suggests knitr, rmarkdown, stringr, sarp.snowprofile.alignment

VignetteBuilder knitr

NeedsCompilation no

Author Pascal Haegeli [aut, cre],
Simon Horton [aut],
Florian Herla [aut],
SFU Avalanche Research Program [fnd]

Maintainer Pascal Haegeli <pascal_haegeli@sfu.ca>

Repository CRAN

Date/Publication 2022-02-10 00:20:02 UTC

R topics documented:

char2numAspect	3
char2numHHI	4
computeRTA	5
computeTSA	6
deriveDatetag	7
export.snowprofileCsv	8
findPWL	9
format_snowprofileLayers	12
getColoursDensity	12
getColoursGrainSize	13
getColoursGrainType	14
getColoursHardness	15
getColoursLWC	16
getColoursSnowTemp	17
grainDict	18
hasUnobservedBasalLayer	18
importRDefaultPackages	19
insertUnobservedBasalLayer	19
is.snowprofile	20
is.snowprofileInstabilitySigns	20
is.snowprofileLayers	21
is.snowprofileSet	21
is.snowprofileTests	22
new_snowprofile	22
numberOfPWLsPerVerticalLevel	24
plot.snowprofile	25
plot.snowprofileSet	27
print.snowprofile	30
rbind.snowprofile	31
rbind.snowprofileSet	32
readSmet	33
reformat_snowprofile	33
scanProfileDates	34
sd_sample_uncorrected	35
setColoursGrainType	36
simplifyGtypes	37
snowprofile	38
snowprofileCaaml	40
snowprofileCsv	41
snowprofileCsv_advanced	43
snowprofileInstabilitySigns	45
snowprofileLayers	46
snowprofilePrf	50
snowprofilePro	51
snowprofileSet	53
snowprofileSno	53

snowprofileTests	54
SPgroup	56
SPmalformatted	56
SPpairs	57
SPtimeline	57
summary.snowprofile	58
summary.snowprofileSet	59
swisscode	60
validate_snowprofile	60
validate_snowprofileLayers	61
writeSmet	62
[.snowprofileSet	63

Index 64

char2numAspect	<i>Conversion of character Aspects to numeric Aspects</i>
----------------	---

Description

Convert character aspects (of snow profile locations) to numeric values. For example, Aspect "N" (north) becomes 0 degrees azimuth.

Usage

```
char2numAspect(charAspect)
```

Arguments

charAspect Character string of aspect location, i.e., one of

- c("N", "NE", "NNE", "ENE", "E", "ESE", "SE", "SSE", "S", "SSW", "SW", "WSW", "W", "WNW", "NW")

Value

Float value of numeric aspect location, North = 0 degree, S = 180 degree

Author(s)

fherla

Examples

```
char2numAspect("W")
char2numAspect("WNW")
```

```
char2numAspect(c("N", NA, "NA", "NE"))
```

`char2numHHI`*Conversion of Hand Hardness Index (HHI)*

Description

Convert character hand hardness index (HHI) of snow layers to numeric values. For example, hand hardness Fist becomes 1, Ice becomes 6.

Usage

```
char2numHHI(charHHI)
```

Arguments

`charHHI` Character string of hand hardness level, i.e., one of

- Fist 'F', 4 Fingers '4F', 1 Finger '1F', Pencil 'P', Knife 'K', or Ice 'I'
- intermediate values allowed, e.g. 'F+', '1F-', 'F-4F'

Value

Float value of numeric hand hardness level between 1 and 6.

Author(s)

fherla

Examples

```
char2numHHI('F+')
char2numHHI('F-')
char2numHHI('F-4F')

## not meaningful:
this_throws_error <- TRUE
if (!this_throws_error) {
  char2numHHI('F-P')
}
```

`computeRTA`*Compute Relative Threshold Sum approach (RTA)*

Description

Monti, F., & Schweizer, J. (2013). A relative difference approach to detect potential weak layers within a snow profile. Proceedings of the 2013 International Snow Science Workshop, Grenoble, France, 339–343. Retrieved from <https://arc.lib.montana.edu/snow-science/item.php?id=1861>

Usage

```
computeRTA(profile, target = c("interface", "layer"))
```

Arguments

<code>profile</code>	a snowprofile object, whose layer properties are entirely known (i.e., no NAs in <code>gtype</code> , <code>hardness</code> , <code>gsize</code> allowed!)
<code>target</code>	Do you want to compute the index for the layers or for the layer interfaces? defaults to both.

Value

the `snowprofile` object is returned with new layer properties describing the index. The index ranges between [0, 1], whereas values > 0.8 indicate structurally unstable layers/interfaces.

Author(s)

fherla

See Also

[computeTSA](#)

Examples

```
sp <- computeRTA(SPpairs$B_modeled1)
plot(sp, TempProfile = FALSE, main = "RTA")
lines(sp$layers$rtax*5, sp$layers$height - 0.5*sp$layers$thickness, type = "b", xlim = c(0, 5))
lines(sp$layers$rtax_interface*5, sp$layers$height, type = "b", xlim = c(0, 5), col = "red")
abline(h = sp$layers$height, lty = "dotted", col = "grey")
abline(v = 0.8*5, lty = "dashed")
```

computeTSA	<i>Compute Threshold Sum Approach (TSA, lemons, yellow flags, 'Nieten')</i>
------------	---

Description

This routine computes the traditional lemons (german 'Nieten') based on absolute thresholds. Since the thresholds are defined in Monti (2014) with different thresholds for manual versus observed profiles, this routine switches between the appropriate thresholds based on the \$type field of the input profile. While manual and whiteboard profiles get one set of thresholds, modeled, vstation, and aggregate type profiles get another set.

Usage

```
computeTSA(profile, target = c("interface", "layer"))
```

Arguments

profile	a snowprofile object
target	Do you want to compute the index for the layers or for the layer interfaces? defaults to both.

Value

the snowprofile object is returned with new layer properties describing the threshold sums. The TSA sums up to 6 indicators, whereas ≥ 5 indicators indicate structurally unstable layers/interfaces.

Author(s)

fherla

References

Schweizer, J., & Jamieson, J. B. (2007). A threshold sum approach to stability evaluation of manual snow profiles. *Cold Regions Science and Technology*, 47(1–2), 50–59. <https://doi.org/10.1016/j.coldregions.2006.08.011>

Monti, F., Schweizer, J., & Fierz, C. (2014). Hardness estimation and weak layer detection in simulated snow stratigraphy. *Cold Regions Science and Technology*, 103, 82–90. <https://doi.org/10.1016/j.coldregions.2014.03.00>

See Also

[computeRTA](#)

Examples

```
sp <- computeTSA(Sppairs$B_modeled1)
plot(sp, TempProfile = FALSE, main = "TSA")
lines(sp$layers$tssa/6*5,
      sp$layers$height - 0.5*sp$layers$thickness, type = "b", xlim = c(0, 5))
lines(sp$layers$tssa_interface/6*5, sp$layers$height, type = "b", xlim = c(0, 5), col = "red")
abline(h = sp$layers$height, lty = "dotted", col = "grey")
abline(v = 5/6*5, lty = "dashed")
```

deriveDatetag

Derive datetag from deposition dates in simulated profiles

Description

This routine derives the datetags of simulated snow profile layers from deposition dates. Datetags usually are deposition dates for crust layers, and burial dates for other weak layers (e.g., SH, FC). If no datetags can be derived, a datetag column of NAs will nevertheless be added to the snowprofile layers. The routine also adds a bdate column for burial dates that are calculated along the way.

Usage

```
deriveDatetag(x)
```

Arguments

x a [snowprofile](#) or [snowprofileLayers](#) object

Value

The x object will be returned with added columns datetag and bdate (i.e., either as snowprofile, or snowprofileLayers)

Author(s)

fherla

Examples

```
## This is not the most meaningful example, but it nicely illustrates the routine:
print(Sppairs$A_manual)

print(deriveDatetag(Sppairs$A_manual))
```

export.snowprofileCsv *Export or write a snowprofile object to a CSV table*

Description

Export or write a snowprofile object to a CSV table

Usage

```
export.snowprofileCsv(  
  profile,  
  filename = stop("filename must be provided"),  
  sep = ",",  
  export.all = "Layers",  
  variables = NA  
)
```

Arguments

profile	snowprofile object
filename	character string, e.g. 'path/to/file.csv'
sep	csv column separator as character string
export.all	one of TRUE, FALSE, 'Layers': export all variables of the snowprofile object to the csv table? If 'Layers', then all layer variables of the snowprofile will be exported.
variables	A tag-value list of the format, e.g. height = 'height_top', to specify column names of specific variables, to customize column order, and/or to include specific profile meta data if export.all == 'Layers' (e.g. easily include the meta data station_id). Note that the tags of the tag-value list need to correspond to elements of the snowprofile object.

Details

Note that existing files with the specified filename will be **overwritten** without warning!

Value

Writes csv file to disk, no return value in R

Author(s)

fherla

See Also

[snowprofileCsv](#)

Examples

```
## export an entire snowprofile object:

export.snowprofileCsv(SPpairs$A_manual, filename = file.path(tempdir(), 'file.csv'),
                      export.all = TRUE)

## export only the layer properties of a snowprofile object,
# and change the column order with few column names:
# All layer variables will be exported, but the three ones provided in 'variables'
# will be the first three columns of the csv table, and their column names will be changed
# accordingly.

export.snowprofileCsv(SPpairs$A_manual, filename = file.path(tempdir(), 'file.csv'),
                      export.all = 'Layers',
                      variables = list(height = 'height_top', hardness = 'hardness',
                                       gtype = 'gt1'))

## export all layer properties of a snowprofile object plus the station ID:

export.snowprofileCsv(SPpairs$A_manual, filename = file.path(tempdir(), 'file.csv'),
                      export.all = 'Layers', variables = list(station_id = 'station_id'))

## check the content of the exported csv file:
csv_content <- read.csv(file.path(tempdir(), 'file.csv'))
head(csv_content)

## or re-import the csv file as snowprofile object:
csv_snowprofile <- snowprofileCsv(file.path(tempdir(), 'file.csv'))
print(csv_snowprofile)
```

findPWL

Find layers of interest (e.g. PWLs) in snowprofile(Layers)

Description

Find one or more layers of interest, such as persistent weak layers (PWL) in a snowprofile or snowprofileLayers object based on combinations of grain type, datetag, grain size, and stability indices (TSA/ RTA) of the layer. The routine can also be used for searching for crusts (or any other grain types).

Usage

```
findPWL(
  x,
  pwl_gtype = c("SH", "DH"),
```

```

pwl_date = NA,
date_range_earlier = as.difftime(5, units = "days"),
date_range_later = as.difftime(0, units = "days"),
bdate_range_earlier = as.difftime(1, units = "days"),
bdate_range_later = as.difftime(1, units = "days"),
threshold_gtype = pwl_gtype,
threshold_gsize = NA,
threshold_TSA = NA,
threshold_RTA = NA,
threshold_NA = TRUE
)

labelPWL(x, ...)

```

Arguments

x	snowprofile or snowprofileLayers object
pwl_gtype	a vector of grain types of interest
pwl_date	a date of interest given as character ('YYYY-MM-DD') or as POSIXct; set to NA to ignore dates.
date_range_earlier	a difftime object that defines a date search window prior to pwl_date. This date range is applied to ddates (deposition dates), or if these are not available to datetags.
date_range_later	analogous to date_range_earlier
bdate_range_earlier	analogous to date_range_earlier, but applied to bdates (burial dates).
bdate_range_later	analogous to bdate_range_earlier.
threshold_gtype	specific grain types that are only deemed a PWL if they pass a certain threshold (see next parameters)
threshold_gsize	a threshold grain size in order to deem threshold_gtype a PWL; set to NA to ignore grain sizes.
threshold_TSA	a threshold TSA value (see computeTSA) in order to deem threshold_gtype a PWL; set to NA to ignore TSA.
threshold_RTA	a threshold RTA value (see computeRTA) in order to deem threshold_gtype a PWL; set to NA to ignore RTA.
threshold_NA	if the threshold you want to apply to threshold_gtype is NA, do you then want to deem the layer a PWL (i.e., set TRUE) or not (i.e., set to FALSE)?
...	passed on to findPWL

Details

In case date considerations are included in your search: at least one of the two conditions (*b*)*date_range_earlier* **AND** at least one of the two conditions (*b*)*date_range_later* needs to be satisfied to return a given layer. If the input object contains deposition dates (*d*date, mostly in simulated profiles), then the date window is applied to both *d*date and a newly computed *b*date; otherwise the date window is applied to the *date*tag (mostly for manual profiles).

The `labelPWL` wrapper function is primarily used by `sarp.snowprofile.alignment::averageSP`.

Value

`findPWL`: An index vector of PWLs that match the desired requirements

`labelPWL`: The input object with an extra boolean column appended to the layer object, called `$layerOfInterest`.

Functions

- `findPWL`: Find layers of interest (e.g., PWLs) in `snowprofile` or `snowprofileLayers`
- `labelPWL`: Label layers of interest (e.g., weak layers) in `snowprofile`

Author(s)

fherla

Examples

```
## get index vector:
findPWL(SPpairs$A_modeled)

## get layers subset:
SPpairs$A_manual$layers[findPWL(SPpairs$A_manual), ]
SPpairs$A_manual$layers[findPWL(SPpairs$A_manual, threshold_gsize = 2.2,
                                threshold_gtype = c("FC", "FCxr")), ]
## all (SH, DH), and (FC, FCxr) >= 1 mm grain size:
SPpairs$A_modeled$layers[findPWL(SPpairs$A_modeled, pwl_gtype = c("SH", "DH", "FC", "FCxr"),
                                threshold_gsize = 1, threshold_gtype = c("FC", "FCxr")), ]
## use TSA threshold:
SPpairs$A_modeled <- computeTSA(SPpairs$A_modeled)
SPpairs$A_modeled$layers[findPWL(SPpairs$A_modeled, pwl_gtype = c("SH", "DH", "FC", "FCxr"),
                                threshold_TSA = 4, threshold_gtype = c("FC", "FCxr")), ]

## searching for a specific pwl_date:
SPpairs$A_manual$layers[findPWL(SPpairs$A_manual, pwl_date = "2019-02-18"), ]
## Why is there a layer included without date information?!
## --> because the layers 'bdate' can be computed from the known 'ddate' of the layer above
## and the resulting 'bdate' lies within the allowed 'date_range'...

## can also be used to search for crusts:
SPpairs$A_manual$layers[findPWL(SPpairs$A_manual, pwl_gtype = "MFcr"), ]
```

```
format_snowprofileLayers
      Format snowprofileLayers
```

Description

Calculate missing data.frame columns based on the given ones, if possible.

Usage

```
format_snowprofileLayers(
  obj,
  target = "all",
  hs = NA,
  maxObservedDepth = NA,
  validate = TRUE,
  dropNAs = TRUE
)
```

Arguments

obj	snowprofileLayers object
target	string, indicating which fields are auto-filled ('all', 'height', 'depth', 'thickness', 'none')
hs	total snow height (cm) if not deductible from given fields
maxObservedDepth	the observed depth of the profile from the snow surface downwards. Will only be used, if no height or thickness exist in obj, or if hs is not given.
validate	Validate obj with validate_snowprofileLayers?
dropNAs	Do you want to drop all columns consisting of NAs only?

Value

copy of obj with auto-filled columns

```
getColoursDensity      Gets colours for plotting snow density values
```

Description

Gets colours for plotting snow density values in snowprofiles. Colours are consistent with niViz at <https://niviz.org>

Usage

```
getColoursDensity(Values, Resolution = 101, Verbose = FALSE)
```

Arguments

Values	Density values (kg/m3)
Resolution	Resolution of colour scale. Default is 100.
Verbose	Switch for writing out value and html colour tuples for debugging.

Value

Array with HTML colour codes

Author(s)

phaegeli

See Also

[getColoursGrainSize](#), [getColoursGrainType](#), [getColoursHardness](#), [getColoursLWC](#), [getColoursSnowTemp](#)

Examples

```
Density <- seq(0,700, by=10)  
plot(x = rep(1,length(Density)), y = Density, col = getColoursDensity(Density), pch = 19, cex = 3)
```

getColoursGrainSize *Gets colours for plotting grain size values*

Description

Gets colours for plotting grain size values in snowprofiles. Colours are consistent with niViz at <https://niviz.org>

Usage

```
getColoursGrainSize(Values, Resolution = 101, Verbose = FALSE)
```

Arguments

Values	Liquid water content values
Resolution	Resolution of colour scale. Default is 100.
Verbose	Switch for writing out value and html colour tuples for debugging.

Value

Array with HTML colour codes

Author(s)

phaegeli

See Also

[getColoursDensity](#), [getColoursGrainType](#), [getColoursHardness](#), [getColoursLWC](#), [getColoursSnowTemp](#)

Examples

```
GrainSize <- seq(0,6, by=0.1)
plot(x = rep(1,length(GrainSize)), y = GrainSize,
     col = getColoursGrainSize(GrainSize), pch = 19, cex = 3)
```

getColoursGrainType *Gets colours for plotting snow grain types*

Description

Grain colours are defined in the grainDict data.frame and the definitions can be changed with setColoursGrainType

Usage

```
getColoursGrainType(Grains, grainDict. = grainDict)
```

Arguments

Grains	grain type (character or list of characters)
grainDict.	lookup table to use. Note, the easiest and best way to do this is via setColoursGrainType. This input variable here is only a hack to change the grainDict explicitly when calling plot.snowprofile via Col, and beforehand computing Col = Col <-sapply(Profile\$layers\$getColoursGrainType(x,grainDict = setColoursGrainType('sarp-reduced'))); This is only necessary in specific environments (e.g. a shiny app)

Value

Array with HTML colour codes

Author(s)

phaegeli, shorton, fherla

See Also

[setColoursGrainType](#), [getColoursDensity](#), [getColoursGrainSize](#), [getColoursHardness](#), [getColoursLWC](#), [getColoursSnowTemp](#)

Examples

```
Grains <- c('PP', 'DF', 'RG', 'FC', 'FCxr', 'DH', 'SH', 'MF', 'MFcr', 'IF')
Colours <- getColoursGrainType(Grains)
Colours

plot(1:length(Grains), col = Colours, pch = 20, cex = 3)
text(1:length(Grains), 1:length(Grains), Grains, pos = 1)
```

getColoursHardness *Gets colours for plotting snow hardness values*

Description

Gets colours for plotting snow hardness values in snowprofiles.

Usage

```
getColoursHardness(Values, Resolution = 101, Verbose = FALSE)
```

Arguments

Values	Hardness values
Resolution	Resolution of colour scale. Default is 100.
Verbose	Switch for writing out value and html colour tuplets for debugging.

Value

Array with HTML colour codes

Author(s)

phaegeli

See Also

[getColoursDensity](#), [getColoursGrainSize](#), [getColoursGrainType](#), [getColoursLWC](#), [getColoursSnowTemp](#)

Examples

```
Hardness <- c(1:5)
plot(x = rep(1,length(Hardness)), y = Hardness,
     col = getColoursHardness(Hardness), pch = 19,cex = 3)
```

getColoursLWC

Gets colours for plotting LWC values

Description

Gets colours for plotting LWC values in snowprofiles. Colours are consistent with niViz at <https://niviz.org>

Usage

```
getColoursLWC(Values, Resolution = 101, Verbose = FALSE)
```

Arguments

Values	Liquid water content values
Resolution	Resolution of colour scale. Default is 100.
Verbose	Switch for writing out value and html colour tuples for debugging.

Value

Array with HTML colour codes

Author(s)

phaegeli

See Also

[getColoursDensity](#), [getColoursGrainSize](#), [getColoursGrainType](#), [getColoursHardness](#), [getColoursSnowTemp](#)

Examples

```
LWC <- seq(0,6, by = 0.1)
plot(x = rep(1,length(LWC)), y = LWC, col = getColoursLWC(LWC), pch = 19, cex = 3)
```

getColoursSnowTemp *Gets colours for plotting snow temperature values*

Description

Gets colours for plotting snow temperature values in snowprofiles. Colours are consistent with niViz at <https://niviz.org>

Usage

```
getColoursSnowTemp(Values, Resolution = 101, Verbose = FALSE)
```

Arguments

Values	Snow temperature values
Resolution	Resolution of colour scale. Default is 100.
Verbose	Switch for writing out value and html colour tuples for debugging.

Value

Array with HTML colour codes

Author(s)

phaegeli

See Also

[getColoursDensity](#), [getColoursGrainSize](#), [getColoursGrainType](#), [getColoursHardness](#), [getColoursLWC](#)

Examples

```
SnowTemp <- c(-25:0)
plot(x = rep(1,length(SnowTemp)), y = SnowTemp,
     col = getColoursSnowTemp(SnowTemp), pch = 19,cex = 3)
```

grainDict	<i>A data.frame storing the grain type colours</i>
-----------	--

Description

The colours can be changed by calling the function [setColoursGrainType](#), see examples below.

Usage

```
grainDict
```

Format

A data.frame

Examples

```
print(grainDict)

## change colours for subsequent plots:
grainDict <- setColoursGrainType('sarp-reduced')
```

hasUnobservedBasalLayer	<i>Check whether a profile is observed down to ground or not</i>
-------------------------	--

Description

Check whether a profile is observed down to ground or not

Usage

```
hasUnobservedBasalLayer(x)
```

Arguments

x a [snowprofile](#), or [snowprofileLayers](#) object

Value

boolean TRUE/FALSE

```
importRDefaultPackages
  Import R_DEFAULT_PACKAGES
```

Description

Import R_DEFAULT_PACKAGES

Usage

```
importRDefaultPackages()
```

```
insertUnobservedBasalLayer
```

Insert a special layer at the bottom to indicate a snow profile that's unobserved from a specific point down to the ground internal function, not exported. used in snowprofileLayers

Description

Insert a special layer at the bottom to indicate a snow profile that's unobserved from a specific point down to the ground internal function, not exported. used in snowprofileLayers

Usage

```
insertUnobservedBasalLayer(object, basal_offset, setBasalThicknessNA = FALSE)
```

Arguments

object	snowprofileLayers object
basal_offset	a positive numeric scalar indicating the thickness of the basal unobserved layer(s)
setBasalThicknessNA	boolean TRUE/FALSE indicating whether the thickness of the inserted layer should be basal_offset or NA. Setting the thickness to NA corresponds to setting a flag that the depth of the profile (i.e., the unobserved basal layers) is unknown. This often happens in manual profiles which only observe the uppermost meter (or so) of the snowpack

Value

same object with basal layer inserted as individual row in the data.frame

Author(s)

fherla

is.snowprofile *Check class snowprofile*

Description

Check if object is of class [snowprofile](#)

Usage

```
is.snowprofile(x)
```

Arguments

x object to test

Value

boolean

is.snowprofileInstabilitySigns
Check class snowprofileInstabilitySigns

Description

Check if object is of class [snowprofileInstabilitySigns](#)

Usage

```
is.snowprofileInstabilitySigns(x)
```

Arguments

x object to test

Value

boolean

is.snowprofileLayers *Check class snowprofileLayers*

Description

Check if object is of class snowprofileLayers

Usage

is.snowprofileLayers(x)

Arguments

x object to test

Value

boolean

is.snowprofileSet *Check class snowprofileSet*

Description

Check if object is of class [snowprofileSet](#)

Usage

is.snowprofileSet(x)

Arguments

x object to test

Value

boolean

is.snowprofileTests *Check class snowprofileTests*

Description

Check if object is of class [snowprofileTests](#)

Usage

```
is.snowprofileTests(x)
```

Arguments

x object to test

Value

boolean

new_snowprofile *Low-level constructor function for a snowprofile object*

Description

Low-cost, efficient constructor function to be used by users who know what they're doing. If that's not you, use the high-level constructor [snowprofile](#).

Usage

```
new_snowprofile(  
  station = character(),  
  station_id = character(),  
  datetime = as.POSIXct(NA),  
  latlon = as.double(c(NA, NA)),  
  elev = double(),  
  angle = double(),  
  aspect = double(),  
  hs = double(),  
  maxObservedDepth = double(),  
  type = character(),  
  band = character(),  
  zone = character(),  
  comment = character(),  
  hn24 = double(),  
  hn72 = double(),  
  ski_pen = double(),
```

```

    layers = snowprofileLayers(),
    tests = snowprofileTests(),
    instabilitySigns = snowprofileInstabilitySigns()
  )

```

Arguments

station	character string
station_id	character string
datetime	date and time as class POSIXct in most meaningful timezone (timezone can be converted very easily: e.g. print(profile\$datetime, tz = 'EST').
latlon	2-element vector latitude (first), longitude (second)
elev	profile elevation (m)
angle	slope angle (degree)
aspect	slope aspect (degree)
hs	total snow height (cm); if not provided, the field will be derived from the profile layers.
maxObservedDepth	equivalent to hs for full profiles that go down to the ground. for test profiles that only observe the upper part of the snowpack this value refers to the maximum depth of the profile observation.
type	character string, must be either 'manual', 'modeled', 'vstation', 'aggregate', or 'whiteboard'
band	character string describing elevation band as ALP, TL, BTL (alpine, treeline, below treeline)
zone	character string describing the zone or region of the profile location (e.g., BURN-ABY_MTN)
comment	character string with any text comments
hn24	height of new snow within 24 h
hn72	height of new snow within 72 h
ski_pen	skier penetration depth (cm)
layers	snowprofileLayers object
tests	snowprofileTests object
instabilitySigns	snowprofileInstabilitySigns object

Value

snowprofile object

`numberOfPWLSPerVerticalLevel`*Count number of PWLS per vertical level*

Description

This is a wrapper function to bin several weak layers (or crusts) into vertical levels. The layers to be binned can be controlled with a provided index vector for full customization.

Usage

```
numberOfPWLSPerVerticalLevel(x, pwl_idx, depth_breaks = c(0, 30, 80, 150, Inf))
```

Arguments

<code>x</code>	<code>snowprofile</code> or <code>snowprofileLayers</code> object
<code>pwl_idx</code>	an index vector that corresponds to the layers of interest. Tip: this can also be a call to <code>findPWL</code> , see examples.
<code>depth_breaks</code>	a vector of break points referring to absolute depth values. <code>Inf</code> is a placeholder for max depth.

Value

This function returns a table object

Author(s)

fherla

Examples

```
SH_idx <- findPWL(Sppairs$C_day1, pwl_gtype = "SH")
numberOfPWLSPerVerticalLevel(Sppairs$C_day1, SH_idx)

numberOfPWLSPerVerticalLevel(Sppairs$C_day2, findPWL(Sppairs$C_day2))
```

plot.snowprofile	<i>Plot hardness profile</i>
------------------	------------------------------

Description

Plot hardness profile

Usage

```
## S3 method for class 'snowprofile'
plot(
  x,
  TempProfile = TRUE,
  xlimTemp = NULL,
  Col = "auto",
  TopDown = "auto",
  axes = TRUE,
  xlab = "",
  emphasizeLayers = FALSE,
  emphasis = "95",
  failureLayers = FALSE,
  failureLayers.cex = 1,
  failureLayers.col = "red",
  nYTicks = 4,
  ymax = max(c(x$maxObservedDepth, x$hs), na.rm = TRUE),
  alignWithBottomUpPlot = FALSE,
  highlightUnobservedBasalLayers = TRUE,
  label.datetags = FALSE,
  ...
)
```

Arguments

x	snowprofile object
TempProfile	draw unscaled temperature profile (default = TRUE)? Temperature data needs to be included in the snowprofile object either under x\$layers\$temperature, or in a separate x\$temperatureProfile data.frame providing a vertical grid independent from the snow layers.
xlimTemp	the x limits in degrees Celsius for the temperature profile (if left empty it scales to the range of temperature values)
Col	vector of colours corresponding to the grain types in the profile (defaults to a lookup table)
TopDown	Option to plot by depth instead of height with zero depth on top of plot (default = FALSE)
axes	Should axes be printed?

xlab	x-axis label, defaults to an empty string
emphasizeLayers	index OR character vector (grain types) of layers to be emphasized (i.e. all other layers become slightly transparent)
emphasis	2 digit quoted number between '01'-'99' to control the degree of emphasis; the higher the stronger
failureLayers	height vector of failure layers that will be indicated with an arrow
failureLayers.cex	factor to shrink or enlarge the arrow
failureLayers.col	color of arrow, can also be a vector of same length as failureLayers to color different arrows differently
nYTicks	number of tick marks at yaxis
ymin	the minimum ylim value
ymax	the maximum ylim value
alignWithBottomUpPlot	useful when aligning the yaxis grids of bottom up profileSet plots and top down hardness plots.
highlightUnobservedBasalLayers	draw sine wave at lowest observed layer to highlight unobserved layers below
label.datetags	label the datetags of the snowprofile layers? (Won't produce a pretty plot, but give you some more information for analysis)
...	other parameters to barplot

See Also

[plot.snowprofileSet](#)

Examples

```

plot(Sppairs$A_manual)
plot(Sppairs$A_manual, Col = 'black')
plot(Sppairs$A_manual, emphasizeLayers = c(5, 11),
      failureLayers = Sppairs$A_manual$layers$height[5], failureLayers.cex = 1.5)
plot(Sppairs$A_manual, emphasizeLayers = 'SH')
plot(Sppairs$A_manual, TopDown = TRUE)
plot(Sppairs$A_modeled, TempProfile = TRUE, xlimTemp = c(-30,10))

# highlight unobserved basal layers:
plot(snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
                                           hardness = c(2, 3, 1),
                                           gtype = c('FC', NA, 'PP'),
                                           hs = 70,
                                           maxObservedDepth = 50)), TopDown = TRUE, ymax = 80)

```

plot.snowprofileSet *Plot a single layer property in multiple profiles side-by-side*

Description

A flexible function to plot multiple snowprofiles either in a timeseries or various types of groups.

Usage

```
## S3 method for class 'snowprofileSet'
plot(
  x,
  SortMethod = c("time", "unsorted", "hs", "elev", "presorted"),
  ColParam = c("gtype", "hardness", "temperature", "density", "gsize"),
  TopDown = FALSE,
  DateStart = NA,
  DateEnd = NA,
  Timeseries_labels = c("weekly", "monthly", NA),
  ylim = NULL,
  OutlineLyrs = FALSE,
  OutlineProfile = NULL,
  HorizGrid = TRUE,
  yaxis = TRUE,
  main = NA,
  ylab = NA,
  xlab = NA,
  box = TRUE,
  xticklabels = FALSE,
  xtick.las = 2,
  yPadding = 10,
  xPadding = 0.5,
  hardnessResidual = 1,
  hardnessScale = 1,
  hardnessOffset = -0.5,
  k = NULL,
  offset = as.Date(NA),
  ...
)
```

Arguments

x	An object of class snowprofileSet
SortMethod	How to arrange profiles along the x-axis. Options include timeseries (default = 'time'), in existing order of Profiles list ('unsorted'), sorted by HS ('hs'), or elevation ('elev')
ColParam	What parameter to show with colour. So far the following types are available: graintype (default), hardness, temperature, density, grainsize.

TopDown	Option to plot by depth instead of height with zero depth on top of plot (default = FALSE)
DateStart	Start date for timeseries plots (SortMethod = 'time'). If not provided, the function takes the date range from Profiles (default = NA).
DateEnd	End date for timeseries plots (SortMethod = 'time'). If not provided, the function takes the date range from Profiles (default = NA).
Timeseries_labels	Label Saturdays "weekly", "monthly", or NA
ylim	Vertical range of plot
OutlineLyr	Switch for outlining layers (default = FALSE)
OutlineProfile	vector of profile indices that will be outlined to highlight them
HorizGrid	Draw horizontal grid at layer heights (default = TRUE)
yaxis	draw a y-axis? (either FALSE, TRUE draws yaxis left, "right" draws yaxis on the right plot side) <i>Note</i> that in case of "right" you need to adjust par(mar = ...), disable ylab and manually draw an xlab with mtext.
main	Main title
ylab	y-axis label; disable ylab by providing an empty string (i.e., ylab = "")
xlab	x-axis label; disable xlab by providing an empty string (i.e., xlab = "")
box	Draw a box around the plot (default = TRUE)
xticklabels	Label the profiles with their "names", "originalIndices" (prior to sorting), "dates", or a custom character array
xtick.las	Orientation of labels if xticklabels is specified.
yPadding	Padding between ylim and limits of data, default = 10. Note that R will still put padding by default. If you want to prohibit that entirely, specify xaxs = 'i', or yaxs = 'i'.
xPadding	Padding between xlim and limits of data, default = 0.5. Note that R will still put padding by default. If you want to prohibit that entirely, specify xaxs = 'i', or yaxs = 'i'.
hardnessResidual	Value within [0, 1] to control the minimum horizontal space of each layer that will be colored irrespective of the layer's hardness. A value of 1 corresponds to no hardness being shown.
hardnessScale	A scaling factor that exaggerates the hardness profile to subsequent cells on the x-axis. Useful for time series of sparse profile observations. Note that this scaling factor is unused when hardnessScale = 1 and that it gets more influential the smaller hardnessScale gets. Also note, that a hardnessScale > 1 can lead to profiles overlapping.
hardnessOffset	offsets the profile location on the x-axis
k	a sorting vector if SortMethod = "presorted".
offset	Provide a Date or POSIXct offset if you want to offset the vertical snow height/depth axis so that the offset date aligns with snow depth/height 0.
...	Additional parameters passed to plot()

Details

The routine allows you to plot colored sequences only, or to include hardness profile information as well. See parameter `hardnessResidual` and the examples for more details. To change the font size of labels etc, use `par()` with the parameters `cex.lab`, `cex.axis`, etc.

Author(s)

shorton, fherla, phaegeli

See Also

[plot.snowprofile](#), [SPgroup](#)

Examples

```
## Standard profile timeline (e.g. https://niviz.org)
plot(SPtimeline)

## Group of profiles with same timestamp
plot(SPgroup, SortMethod = 'unsorted') # sorted in same order as list
plot(SPgroup, SortMethod = 'hs') # sorted by snow height
plot(SPgroup, SortMethod = 'elev') # sorted by elevation

## Colour layers by other properties
plot(SPtimeline, ColParam = 'density')

## Align layers by depth instead of height
plot(SPtimeline, TopDown = TRUE)

## Timelines with specific date ranges
plot(SPtimeline, DateEnd = '2017-12-17')
plot(SPtimeline, DateStart = '2017-12-15', DateEnd = '2017-12-17')

## Show hardness profile, too:
plot(SPtimeline, hardnessResidual = 0.5)

## Additional examples of plot dimensions and labelling
## Label the indices of the profiles in the list:
plot(SPgroup, SortMethod = 'elev', xticklabels = "originalIndices")
## ... and with minimized axis limits and their station ID names:
plot(SPgroup, SortMethod = 'elev', xticklabels = sapply(SPgroup, function(x) x$station_id),
      yPadding = 0, xPadding = 0, xaxs = 'i', yaxs = 'i')
## sorted by depth, and without box:
plot(SPgroup, SortMethod = 'hs', TopDown = TRUE, box = FALSE)

## Apply a date offset to investigate which layers formed around that day of interest:
pwl_exists <- sapply(SPgroup, function(sp)
  {length(findPWL(sp, pwl_date = "2019-01-21", pwl_gtype = c("SH", "DH"),
    date_range_earlier = as.difftime(2, unit = "days")) > 0)})
k <- order(pwl_exists, decreasing = TRUE)
plot(SPgroup, SortMethod = 'presorted', k = k, xticklabels = "originalIndices",
```

```
offset = as.Date("2019-01-21"), xlab = "<-- Jan 21 PWL exists | does not exist -->")
abline(v = max(which(pwl_exists[k]))+0.5, lty = "dashed")
```

`print.snowprofile` *Print snowprofile object*

Description

Print snowprofile object

Usage

```
## S3 method for class 'snowprofile'
print(x, pretty = TRUE, nLayers = NA, ...)
```

Arguments

<code>x</code>	<code>snowprofile</code> object
<code>pretty</code>	pretty print the object (data.frame-like instead of list-like)
<code>nLayers</code>	only print the first few layers (cf., head)
<code>...</code>	passed to print.default

Value

object gets printed to console

Examples

```
## pretty print
SPpairs$A_manual
## or alternatively:
print(SPpairs$A_manual)
## reduce number of layers printed:
print(SPpairs$A_manual, nLayers = 6)

## print profile non-pretty (i.e., like the data is stored):
print(SPpairs$A_manual, pretty = FALSE)
```

rbind.snowprofile	<i>Convert snowprofile into data.frame with columns for metadata</i>
-------------------	--

Description

Convert snowprofile object into data.frame with a row for each layer and additional columns with metadata

Usage

```
## S3 method for class 'snowprofile'  
rbind(..., deparse.level = 1)
```

Arguments

... Object of class [snowprofile](#)
deparse.level Argument for generic rbind method

Details

Metadata columns are calculated with [summary.snowprofile](#)

Value

data.frame

Author(s)

shorton

See Also

[summary.snowprofile](#), [rbind.snowprofileSet](#)

Examples

```
Profile <- SPgroup[[1]]  
ProfileTable <- rbind(Profile)  
head(ProfileTable)
```

rbind.snowprofileSet *Concatenate snowprofileSet into a large data.frame with a row for each layer*

Description

A wrapper to apply [rbind.snowprofile](#) to each profile in a [snowprofileSet](#) then concatenate

Usage

```
## S3 method for class 'snowprofileSet'  
rbind(..., deparse.level = 1)
```

Arguments

... Object of class [snowprofileSet](#)
deparse.level Argument for generic rbind method

Details

Returns a large data.frame with a row for each layer and additional columns with metadata (calculated with [summary.snowprofile](#))

Value

data.frame

Author(s)

shorton

See Also

[summary.snowprofile](#), [rbind.snowprofile](#)

Examples

```
## Create rbind table  
ProfileTable <- rbind(SPgroup)  
head(ProfileTable)  
  
## Filter by layer properties  
SHlayers <- subset(ProfileTable, gtype == 'SH')  
summary(SHlayers)  
plot(elev ~ gsize, SHlayers)
```

readSmet	<i>Parse a SMET file</i>
----------	--------------------------

Description

Read contents of a SMET file https://models.slf.ch/docserver/meteoio/SMET_specifications.pdf

Usage

```
readSmet(Filename)
```

Arguments

Filename Path to a smet file

Value

List containing metadata and data

Author(s)

shorton

See Also

[writeSmet](#), [snowprofileSno](#), [snowprofilePrf](#), [snowprofilePro](#)

Examples

```
## Path to example smet
Filename <- system.file('extdata', 'example.smet', package = 'sarp.snowprofile')
Wx = readSmet(Filename)
str(Wx)
```

reformat_snowprofile	<i>Reformat a malformed snowprofile object</i>
----------------------	--

Description

Reformat a malformed snowprofile object. A malformed object may use field names that deviate from our suggested field names (e.g., `grain_type` instead of `gtype`), or it may use data types that are different than what we suggest to use (e.g., `ddate` as type `Date` instead of `POSIXct`). Basically, if your snowprofile object fails the test of [validate_snowprofile](#) due to the above reason this function should fix it.

Usage

```
reformat_snowprofile(profile, currentFields = NULL, targetFields = NULL)
```

Arguments

```
profile          snowprofile object
currentFields    array of character strings specifying the current field names that you want to
                 change
targetFields     array of same size than currentFields specifying the new field names
```

Examples

```
## check the malformed profile:
this_throws_error <- TRUE
if (!this_throws_error) {
  validate_snowprofile(SPmalformatted[[1]])
}
## i.e., we see that elev and ddate are of wrong data type,
## and a warning that grain_type is an unknown layer property.

## reformat field types, but not the field name:
betterProfile <- reformat_snowprofile(SPmalformatted[[1]])
## i.e., no error is raised anymore, but only the grain_type warning

## so let's reformat also the field names:
optimalProfile <- reformat_snowprofile(SPmalformatted[[1]], "grain_type", "gtype")

## reformat a list of profiles with the same configuration:
SPmalformatted_reformatted <- lapply(SPmalformatted, reformat_snowprofile,
                                     currentFields = "grain_type", targetFields = "gtype")

## the malformed profile set finally is correctly formatted:
lapply(SPmalformatted_reformatted, validate_snowprofile)
```

scanProfileDates *Read profile dates from prf/pro file*

Description

Before reading entire SNOWPACK output it can be helpful to scan the profile timestamps first

Usage

```
scanProfileDates(Filename, tz = "UTC")
```

Arguments

Filename filename
 tz time zone (default = 'UTC')

Value

vector of as.POSIXct timestamps

Author(s)

shorton

See Also

[snowprofilePrf](#), [snowprofilePro](#)

Examples

```
## Path to example prf file
Filename <- system.file('extdata', 'example.prf', package = 'sarp.snowprofile')

## Scan dates in file
Dates <- scanProfileDates(Filename)
print(Dates)
```

sd_sample_uncorrected *fast* *uncorrected* *sample* *standard* *deviation*
https://en.wikipedia.org/wiki/Standard_deviation#Rapid_calculation_methods

Description

fast uncorrected sample standard deviation https://en.wikipedia.org/wiki/Standard_deviation#Rapid_calculation_methods

Usage

```
sd_sample_uncorrected(x, xbar = mean(x), na.rm = FALSE)
```

Arguments

x a numeric vector
 xbar arithmetic mean of x
 na.rm remove any NAs before computation of standard deviation?

Value

uncorrected sample standard deviation (i.e., a numeric scalar)

Author(s)

fherla

`setColoursGrainType` *Set colour scale for grain types*

Description

Currently, you can choose between 'iacs', 'iacs2', 'sarp', or 'sarp-reduced'.

Usage`setColoursGrainType(ScaleName)`**Arguments**

ScaleName	Name of graintype colour scale
	<ul style="list-style-type: none">• iacs: scale defined by the <i>International Classification of Seasonal Snow on the Ground</i>• iacs2: scale defined by the <i>International Classification of Seasonal Snow on the Ground</i> with a dark red colour for MFcr layers so that MF and MFcr layers can be better distinguished.• sarp: hazard adjusted colours for grain types based on Horton et al. (2020)• sarp-reduced: hazard adjusted colours for groups of grain types based on Horton et al. (2020)

Value

data.frame containing the new colour values stored in grainDict

References

Horton, S., Nowak, S., and Haegeli, P.: Enhancing the operational value of snowpack models with visualization design principles, *Nat. Hazards Earth Syst. Sci.*, 20, 1557–1572, doi: [10.5194/nhess-2015572020](https://doi.org/10.5194/nhess-2015572020), 2020.

See Also

[grainDict](#), [getColoursGrainType](#)

Examples

```
## Current/default grain type colours
grainDict
plot(SPpairs$A_manual, main = 'Snow profile with default colours')

## Change to IACS colours
grainDict <- setColoursGrainType('IACS')
grainDict
plot(SPpairs$A_manual, main = 'Snow profile with IACS colours')

## Change to IACS colours with adjusted MFcr (darkred)
grainDict <- setColoursGrainType('IACS2')
grainDict
plot(SPpairs$A_manual, main = 'Snow profile with IACS colours and adjusted darkred MFcr')

## Change to SARP colours
grainDict <- setColoursGrainType('SARP')
grainDict
plot(SPpairs$A_manual, main = 'Snow profile with SARP colours')

## Change to reduced SARP colours
grainDict <- setColoursGrainType('SARP-reduced')
grainDict
plot(SPpairs$A_manual, main = 'Snow profile with a reduced set of SARP colours')
```

simplifyGtypes

Simplify detailed grain types to parent classes

Description

The IACS records grain types in major and minor classes, e.g. precipitation particles PP can be subclassified into stellar dendrites PPs_d. Some of these subclasses are not supported in this R package and so this function simplifies the unsupported grain type subclasses into their supported main classes. If a given grain type cannot be simplified, a NA value is returned for it.

Usage

```
simplifyGtypes(gtypes, supported_gtypes = grainDict$gtype)
```

Arguments

`gtypes` an array of character grain types following IACS standards
`supported_gtypes` an array of supported grain types that will determine the simplification

Value

the modified input array

Author(s)

fherla

Examples

```
## create an array of gtypes
gtypes <- c('FCxr', 'RGxf', 'PPsd', 'PP', 'IFrc', "KKfx")

## simplify gtypes to supported_gtypes:
simplifyGtypes(gtypes)
```

snowprofile

High-level constructor for a snowprofile object

Description

Conveniently create a snowprofile object. Calls low-level constructor (only available internally: [new_snowprofile](#)), asserts correctness through a snowprofile validator function ([validate_snowprofile](#)) and yields meaningful error messages. Use low-level constructor if you generate many (!) profiles.

Usage

```
snowprofile(
  station = as.character(NA),
  station_id = as.character(NA),
  datetime = as.POSIXct(NA),
  latlon = as.double(c(NA, NA)),
  elev = as.double(NA),
  angle = as.double(NA),
  aspect = as.double(NA),
  hs = as.double(NA),
  maxObservedDepth = as.double(NA),
  type = "manual",
  band = as.character(NA),
  zone = as.character(NA),
  comment = as.character(NA),
  hn24 = as.double(NA),
  hn72 = as.double(NA),
  ski_pen = as.double(NA),
  layers = snowprofileLayers(dropNAs = FALSE, validate = FALSE),
  tests = snowprofileTests(dropNAs = FALSE),
  instabilitySigns = snowprofileInstabilitySigns(dropNAs = FALSE),
  validate = TRUE,
  dropNAs = TRUE
)
```

Arguments

station	character string
station_id	character string
datetime	date and time as class POSIXct in most meaningful timezone (timezone can be converted very easily: e.g. <code>print(profile\$datetime, tz = 'EST')</code>).
latlon	2-element vector latitude (first), longitude (second)
elev	profile elevation (m)
angle	slope angle (degree)
aspect	slope aspect (degree)
hs	total snow height (cm); if not provided, the field will be derived from the profile layers.
maxObservedDepth	equivalent to hs for full profiles that go down to the ground. for test profiles that only observe the upper part of the snowpack this value refers to the maximum depth of the profile observation.
type	character string, must be either 'manual', 'modeled', 'vstation', 'aggregate', or 'whiteboard'
band	character string describing elevation band as ALP, TL, BTL (alpine, treeline, below treeline)
zone	character string describing the zone or region of the profile location (e.g., BURN-ABY_MTN)
comment	character string with any text comments
hn24	height of new snow within 24 h
hn72	height of new snow within 72 h
ski_pen	skier penetration depth (cm)
layers	snowprofileLayers object
tests	snowprofileTests object
instabilitySigns	snowprofileInstabilitySigns object
validate	Validate the object with validate_snowprofile?
dropNAs	Do you want to drop non-mandatory snowprofile and snowprofileLayers fields that are NA only?

Value

snowprofile object

Author(s)

shorton, fherla

See Also

[summary.snowprofile](#), [plot.snowprofile](#), [snowprofileLayers](#), [snowprofileTests](#), [snowprofileInstabilitySigns](#), [SPpairs](#)

Examples

```
## Empty snowprofile:
snowprofile()

## Test profile:
testProfile <- snowprofile(station = 'SARPstation', station_id = 'SARP007',
                           datetime = as.POSIXct('2019/04/01 10:00:00', tz = 'PDT'),
                           latlon = c(49.277223, -122.915084), aspect = 180,
                           layers = snowprofileLayers(height = c(10, 25, 50),
                                                         hardness = c(3, 2, 1),
                                                         gtype = c('FC', NA, 'PP')))

summary(testProfile)
plot(testProfile)
```

snowprofileCaaml

Read a Caaml file into a snowprofile object

Description

Note, that this function only provides a starting point for loading caaml files into R. Currently, caaml files exported from niviz.org, or snowpilot.org should be compatible with this routine. However, this routine only extracts some metadata and some of the most important layer characteristics. While a temperature profile (that is independent from the layers) is extracted, no other variables that can be written into a caaml file are currently being read (such as stability test results, etc).

Usage

```
snowprofileCaaml(
  caamlFile,
  sourceType = NA,
  readStabilityTests = TRUE,
  validate = TRUE
)
```

Arguments

caamlFile `'path/to/file.caaml'`

sourceType choose `'manual'`, `'modeled'`, `'vstation'`, `'aggregate'` or `'whiteboard'`; while this routine has some functionality built in to detect sourceTypes under certain circumstances, it needs to be provided in most cases.

readStabilityTests
boolean (this is still beta version and can throw errors sometimes)

validate
Should the resulting snowprofile object be validated by [validate_snowprofile?](#)

Details

- There is still a bug related to non-numeric aspects (e.g., E instead of 90).
- The [snowprofileCsv](#) function provides a lot more flexibility to read in data, if you can choose the format of your underlying data. Don't hesitate to reach out though if your caaml files throw errors and you need help! If you extend this routine, please also reach out and let us know, so we can update this package with your code extensions.

Value

snowprofile object

Author(s)

fherla

Examples

```
## load example caaml file that ships with package:
caamlFile <- system.file('extdata', 'example.caaml', package = 'sarp.snowprofile')

## read caaml file:
profile <- snowprofileCaaml(caamlFile, sourceType = 'vstation')

## other file with slightly different xml namespace, structure, etc (including stability test):
caamlFile2 <- system.file('extdata', 'example2.caaml', package = 'sarp.snowprofile')
profile2 <- snowprofileCaaml(caamlFile2, sourceType = 'manual')
```

snowprofileCsv *Read csv file into a snowprofile object*

Description

Read csv file into a snowprofile object

Usage

```
snowprofileCsv(  
  path,  
  header = TRUE,  
  sep = ",",  
  use.swisscode = FALSE,
```

```

    height = "height",
    gtype = "gtype",
    hardness = "hardness",
    ...,
    crust.val = 2
)

```

Arguments

path	'path/to/file.csv'
header	is there a header line in the csv file to explain the column names? If not, specify a character vector of column names in the correct order.
sep	csv column separator as string
use.swisscode	boolean; are grain types given as (numeric) swisscode (TRUE) or as character strings (FALSE)? If TRUE, grain types can be given as three-digit code (gt1 gt2 gt3), or as one-digit code specifying the primary grain type <i>if</i> another column is provided that specifies crusts. See Details and Examples for more information.
height	character string referring to the csv column of the top layer interfaces
gtype	character string referring to the csv column of the grain types
hardness	character string referring to the csv column of the layer hardnesses
...	provide name-value pairs of additional csv columns (in the form gsize = 'csv-GrainSize-ColName'), e.g. <ul style="list-style-type: none"> • profile specific info: station, station_id, datetime, latlon, elev, angle, aspect, type (see snowprofile) • layer specific info: deposition date, grain size, ssi, ... (see snowprofileLayers)
crust.val	If a column 'crust' is provided, what value of 'crust' defines MFcr? Mostly, either 2 (default) or 1. See Details.

Details

The minimum information required to construct a valid [snowprofile](#) object is height, gtype and hardness. Currently, substituting height with a depth vector is not supported.

If profile specific information is provided in the csv table, it can only be included into the snowprofile object through the exact field names (see above). However, layer specific information can be named arbitrarily (except for the three required fields).

Regarding **swisscode**: The SNOWPACK documentation specifies that MFcr are encoded as (gt1|gt2|gt3) = (7|xl|2), i.e. gt1 == 7 and gt3 == 2. This is also how this routine handles the grain type encoding per default. However, some csv tables might be provided using swisscode encoding and providing gt1, gt2, and gt3 as individual one-digit columns. In those cases, gt3 could be defined as a boolean (0 or 1), where gt1 == 7 and gt3 == 1 represent crusts, instead of the aforementioned standard definition of gt1 == 7 and gt3 == 2. To handle these cases, crust.val can be set to 1, instead of its default crust.val = 2.

Value

snowprofile object

Author(s)

fherla

See Also[snowprofileCsv_advanced](#)**Examples**

```
## imagine a csv table with a very straightforward format,
## similar to the following data.frame:
(DF <- data.frame(height = c(50, 80, 100), gtype = c('FC', 'RG', 'PP'), hardness = c(1, 3, 2)))
## write DF to a temporary file:
write.csv(DF, file = file.path(tempdir(), 'file.csv'))

## read this file very easily by
profile <- snowprofileCsv(file.path(tempdir(), 'file.csv'))
profile

## imagine a csv table that requires a bit more customization,
## similar to the following data.frame:
(DF <- data.frame(ID = rep(1234, times = 3), layer_top = c(10.5, 15, 55.0), gt1 = c(5, 7, 2),
                  gs = c(5.0, 1.5, 1.0), crust = c(0, 1, 0), hardness = c('F', 'P', '4F+')))
write.csv(DF, file = file.path(tempdir(), 'file.csv'))

profile <- snowprofileCsv(file.path(tempdir(), 'file.csv'), height = 'layer_top', gtype = 'gt1',
                          use.swisscode = TRUE, gsize = 'gs', crust.val = 1)
profile
## Note that the csv column 'crust', which specifies whether a MF layer is actually
# a MFcr layer, is already named correctly (i.e., 'crust'). If it were named 'freeze-crust',
# we would need to add to the function call: `crust = 'freeze-crust'`.
# Also note, that we need to provide `crust.val = 1`, since we're not using the standard definition
# of swisscode MFcr encoding (see Details).

## let's assume you want to read the csv file and customize some names, e.g. GrainSIZE:
profile <- snowprofileCsv(file.path(tempdir(), 'file.csv'), height = 'layer_top', gtype = 'gt1',
                          use.swisscode = TRUE, GrainSIZE = 'gs')
profile

## Note that generally in a snowprofile object layer properties can be custom named,
# meta information, e.g. station_id, can not! I.e. you need to use the prescribed names.
```

snowprofileCsv_advanced

Read routine for advanced csv tables containing various snowprofile information

Description

This routine reads blocks of snowprofile metadata, layers, tests, and stability signs. Columns contain different variables, rows different observations. While metadata only contains one row, layers, tests, and signs consist of potentially multiple rows. Within each block of information, mind the correct alignment of rows. Missing values (i.e., NA) need to be left blank or called NA. See the examples below including the example file shipped with the package.

Usage

```
snowprofileCsv_advanced(
  csvFile,
  meta = c("uid", "hs", "maxObservedDepth", "comment"),
  layers = c("depth", "height", "gtype", "hardness", "datetag", "gsize", "gtype_sec",
            "layer_comment"),
  tests = c("test", "result", "fract_char", "score", "test_depth", "test_comment"),
  instabilitySigns = c("instabilitySign_type", "instabilitySign_present",
                     "instabilitySign_comment"),
  sep = ",",
  elev.units = "ft"
)
```

Arguments

csvFile	'path/to/file.csv'
meta	column names of block metadata
layers	column names of block snowprofileLayers
tests	column names of block snowprofileTests
instabilitySigns	column names of block snowprofileInstabilitySigns
sep	csv column separator
elev.units	if set to "ft", the routine will convert to "m". Set to "m" (or anything else) if it should be unchanged

Author(s)

fherla

Examples

```
## load example csv file that ships with package:
csvFile <- system.file('extdata', 'example_adv.csv', package = 'sarp.snowprofile')

profile <- snowprofileCsv_advanced(csvFile, meta = c("uid", "hs", "maxObservedDepth", "comment",
          "datetime", "zone", "station",
          "station_id", "aspect", "elev", "angle"))

plot(profile)
```

 snowprofileInstabilitySigns

Constructor for a snowprofileInstabilitySigns object

Description

Create a snowprofileInstabilitySigns object. Instability signs can for example be whumpfs, cracking, natural avalanches, skier accidental release, ski cutting, etc. For more information, see Canadian Avalanche Association. (2016). Observation Guidelines and Recording Standards for Weather, Snowpack, and Avalanches. Revelstoke, BC, Canada.

Usage

```
snowprofileInstabilitySigns(
  signsFrame = data.frame(type = as.character(NA), present = as.character(NA), comment
    = as.character(NA)),
  dropNAs = TRUE
)
```

Arguments

signsFrame	<p>a data.frame listing snowpack stability signs. Rows correspond to individual observations of instability signs and columns describe at least the fields c("type", "present").</p> <ul style="list-style-type: none"> • type: Sc, Sa, Na, whumpf, crack, ... • present: Was the instability sign present (TRUE), not present (FALSE), or unknown (NA), for example <ul style="list-style-type: none"> – natural avalanches occurred (i.e., Na TRUE), did not occur (i.e., Na FALSE), no observations were carried out (i.e., Na NA) – skiing the slope led to an avalanche (i.e., Sa TRUE) – ski cutting did not release avalanche (i.e., Sc FALSE) – etc
dropNAs	Should empty, non-mandatory columns be dropped from the final snowprofileInstabilitySigns object?

Details

Note: This class might be a temporary solution to digitize instability signs observed in proximity to snowprofiles. The information contained here, might be ported to a more general field observations class that is both independent from snowprofile objects and that is more in line with existing field observation standards.

Value

snowprofileInstabilitySigns object

Author(s)

fherla

See Also[snowprofile](#), [snowprofileLayers](#), [snowprofileTests](#)**Examples**

```
## create a data.frame with instability sign observations
(signsFrame <- data.frame(type = c("Na", "whumpf", "cracking", "Sa"),
                          present = c(FALSE, TRUE, FALSE, FALSE)))

## create snowprofileInstabilitySigns object
instabilitySigns <- snowprofileInstabilitySigns(signsFrame)

## create snowprofile object containing instability signs and check resulting object:
snowprofile(instabilitySigns = instabilitySigns)
```

snowprofileLayers *Constructor for a snowprofileLayers object*

Description

Helper function to conveniently create a snowprofileLayers object, i.e. data.frame with mandatory column fields height (or depth) that provides vertical position of layers. Layers need to be ordered in a sequential manner, and the routine will rearrange the layers so that the last row of the resulting dataframe corresponds to the snow surface. If the vertical location of the layers is given by depth, make sure to provide hs if it's known. Otherwise, provide the field maxObservedDepth or layer thicknesses. Providing only depth will issue a warning and set the corresponding lowest layer thickness to NA. The resulting dataframe will contain all three fields height, depth, and thickness, which will be auto-filled if not provided (see [format_snowprofileLayers](#)). If the columns that describe layer properties are not of equal lengths, their values will be recycled (default data.frame mechanism). Instead of individual layer characteristics, a data.frame can be provided, which will be converted into a snowprofileLayers class. The constructor asserts correctness of the layers object by a call to [validate_snowprofileLayers](#).

Usage

```
snowprofileLayers(
  height = as.double(NA),
  temperature = as.double(NA),
  density = as.double(NA),
  lwc = as.double(NA),
  gsize = as.double(NA),
  gsize_max = as.double(NA),
  gsize_avg = as.double(NA),
```

```

  gtype = as.factor(NA),
  gtype_sec = as.factor(NA),
  hardness = as.double(NA),
  ddate = as.POSIXct(NA),
  bdate = as.Date(NA),
  datetag = as.POSIXct(NA),
  ssi = as.double(NA),
  sphericity = as.double(NA),
  v_strain_rate = as.double(NA),
  crit_cut_length = as.double(NA),
  tsa = as.double(NA),
  tsa_interface = as.double(NA),
  rta = as.double(NA),
  rta_interface = as.double(NA),
  layerOfInterest = as.logical(NA),
  comment = as.character(NA),
  ...,
  hs = as.double(NA),
  maxObservedDepth = as.double(NA),
  layerFrame = NA,
  validate = TRUE,
  dropNAs = TRUE
)

```

Arguments

height	height vector (cm) referring to the top layer interface. Instead of height, depth can also be given and should be accompanied by an array specifying the thickness of the layers, or alternatively, the total snow depth <code>hs</code> and/or the maximum observed depth <code>maxObservedDepth</code> should be provided. Note, that also the depth refers to the top layer interface. See examples!
temperature	snow temperature (deg C)
density	layer density (kg/m3)
lwc	liquid water content (%)
gsize	grain size (mm)
gsize_max	maximum grain size (mm)
gsize_avg	average grain size (mm)
gtype	grain type (character or factor)
gtype_sec	secondary grain type (character or factor)
hardness	numeric hand hardness (use char2numHHI to convert from character hardness)
ddate	deposition date of layer (POSIXct format)
bdate	burial date of layer (Date format)
datetag	of layer (i.e., usually corresponds to <code>ddate</code> for 'MFcr', and to <code>bdate</code> for all other grain types.)
ssi	snow stability index (numeric)

sphericity	between 0 and 1
v_strain_rate	viscous deformation rate (s^{-1})
crit_cut_length	critical crack length (m)
tsa	threshold sum approach for structural instability (also called lemons); valid for the layer, i.e., the weakest interface adjacent to the layer. see computeTSA .
tsa_interface	same as tsa, but valid for top interface of corresponding layer
rta	relative threshold sum approach (following Monti et al 2013, ISSW paper); valid for the layer, i.e., the weakest interface adjacent to the layer. see computeRTA .
rta_interface	same as rta, but valid for top interface of corresponding layer
layerOfInterest	a boolean column to label specific layers of interest, e.g. weak layers. see labelPWL .
comment	character string
...	columns to include in the layers object. Note, that they need to correspond to the according height/depth array. e.g. hardness (can use character hardness or numeric hardness via char2numHHI), ddate (class POSIX), bdate (class Date) gtype (character or factor), density, temperature, gsize, lwc, gsize_max, gtype_sec, ssi, depth, thickness
hs	total snow height (cm), if not deductible from height vector. Particularly important when only a depth grid is provided!
maxObservedDepth	the observed depth of the profile from the snow surface downwards. Will only be used, if no height, thickness, or hs is given.
layerFrame	a data.frame that's converted to a snowprofileLayers class if no other layer characteristics are provided
validate	Validate obj with validate_snowprofileLayers?
dropNAs	Do you want to drop all columns consisting of NAs only?

Value

snowprofileLayers object as data.frame with strings as factors

Author(s)

shorton, fherla

See Also

[snowprofile](#)

Examples

```

## Empty layers object:
snowprofileLayers()

## simple layers example that recycles the hardness 1F+: with warning issued!
snowprofileLayers(height = c(10, 25, 50),
                  hardness = char2numHHI('1F+'),
                  gtype = c('FC', NA, 'PP'))

## create snowprofileLayers object from data.frame
## and feed it into a snowprofile object:
df <- data.frame(height = c(10, 25, 50),
                 hardness = c(2, 3, 1),
                 gtype = c('FC', NA, 'PP'),
                 stringsAsFactors = TRUE)

spL <- snowprofileLayers(layerFrame = df)
(sp <- snowprofile(layers = spL))

##### Create top-down recorded snowprofileLayers #####
## check out how the fields 'hs' and 'maxObservedDepth' are auto-filled in the
## resulting snowprofile object!
## 1.) Specify depth and hs:
## In that case the routine will assume that the deepest layer extends down to the ground
(sp1 <- snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
                                             hardness = c(2, 3, 1),
                                             gtype = c('FC', NA, 'PP'),
                                             hs = 50)))

## note that sp and sp1 are the same profiles:
all(sapply(names(sp$layers), function(cols) {sp$layers[cols] == sp1$layers[cols]}), na.rm = TRUE)

## 2.) Specify depth, hs and thickness or maxObservedDepth:
## This will include a basal layer of NAs to fill the unobserved space down to the ground.
(sp2 <- snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
                                             hardness = c(2, 3, 1),
                                             gtype = c('FC', NA, 'PP'),
                                             hs = 70,
                                             maxObservedDepth = 50)))

## 3.) Specify depth and maxObservedDepth:
## This will include a basal layer of NAs which is 1 cm thick to flag the unknown basal layers.
(sp3 <- snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
                                             hardness = c(2, 3, 1),
                                             gtype = c('FC', NA, 'PP'),
                                             gsize = c(2, NA, NA),
                                             maxObservedDepth = 50)))

## 4.) Specify depth and thickness:

```

```

## This is equivalent to the example spl3 above!
## This will include a basal layer of NAs which is 1 cm thick to flag the unknown basal layers.
(sp4 <- snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
      thickness = c(10, 15, 25),
      hardness = c(2, 3, 1),
      gtype = c('FC', NA, 'PP'))))

## 5.) Specify only depth: issues warning!
(sp5 <- snowprofile(layers = snowprofileLayers(depth = c(40, 25, 0),
      hardness = c(2, 3, 1),
      gtype = c('FC', NA, 'PP'))))

## plot all 5 top.down-recorded profiles:
set <- snowprofileSet(list(sp1, sp2, sp3, sp4, sp5))
plot(set, SortMethod = "unsorted", xticklabels = "originalIndices",
      hardnessResidual = 0.1, hardnessScale = 1.5, TopDown = TRUE,
      main = "TopDown Plot")

plot(set, SortMethod = "unsorted", xticklabels = "originalIndices",
      hardnessResidual = 0.1, hardnessScale = 1.5, TopDown = FALSE,
      main = "BottomUp Plot")

```

snowprofilePrf

Construct snowprofile object from PRF file

Description

Read .prf files from SNOWPACK model output

Usage

```
snowprofilePrf(Filename, ProfileDate = NA, tz = "UTC")
```

Arguments

Filename	path to prf file
ProfileDate	read a single profile from file (default = NA will read all profiles)
tz	time zone (default = 'UTC')

Details

Several SNOWPACK model output formats exist see [SNOWPACK documentation](#)

Definitions of PRF files are provided at https://models.slf.ch/docserver/snowpack/html/prf_format.html

PRF files typically contain profiles from the same station at multiple time steps. If a specific ProfileDate is provided a single snowprofile object is returned (search available dates with scanProfileDates), otherwise all profiles are read and a list of snowprofile objects is returned.

Value

a single snowprofile object or list of multiple snowprofile objects

Author(s)

shorton

See Also

[snowprofilePro](#), [scanProfileDates](#), [snowprofileSno](#)

Examples

```
## Path to example prf file
Filename <- system.file('extdata', 'example.prf', package = 'sarp.snowprofile')

## Scan dates in file
Dates <- scanProfileDates(Filename)
print(Dates)

## Read a single profile by date and plot
ProfileDate <- Dates[3]
Profile <- snowprofilePrf(Filename, ProfileDate = ProfileDate)
plot(Profile)

## Read entire time series and plot
Profiles <- snowprofilePrf(Filename)
plot(Profiles, main = 'Timeseries read from example.prf')
```

snowprofilePro

Construct snowprofile object from PRO file

Description

Read .pro files from SNOWPACK model output

Usage

```
snowprofilePro(Filename, ProfileDate = NA, tz = "UTC")
```

Arguments

Filename	path to pro file
ProfileDate	read a single profile from file (default = NA will read all profiles)
tz	time zone (default = 'UTC')

Details

Several SNOWPACK model output formats exist see [SNOWPACK documentation](#)

Definitions of PRO files are provided at https://models.slf.ch/docserver/snowpack/html/pro_format.html and an example file is available at [niViz](#)

PRO files typically contain profiles from the same station at multiple time steps. If a specific ProfileDate is provided a single snowprofile object is returned (search available dates with scanProfileDates), otherwise all profiles are read and a list of snowprofile objects is returned.

Value

a single snowprofile object or list of multiple snowprofile objects

Author(s)

shorton

See Also

[snowprofilePrf](#), [scanProfileDates](#), [snowprofileSno](#)

Examples

```
## Path to example pro file
Filename <- system.file('extdata', 'example.pro', package = 'sarp.snowprofile')

## Download example pro file from niViz
#Filename <- tempfile(fileext = '.pro')
#download.file('https://niviz.org/resources/example.pro', Filename)

## Scan dates in file
Dates <- scanProfileDates(Filename)
print(Dates)

## Read a single profile by date and plot
ProfileDate <- Dates[3]
Profile <- snowprofilePro(Filename, ProfileDate = ProfileDate)
plot(Profile)

## Read entire time series and plot
Profiles <- snowprofilePro(Filename)
plot(Profiles, main = 'Timeseries read from example.pro')
```

snowprofileSet	<i>Constructor for class snowprofileSet</i>
----------------	---

Description

Constructor for class snowprofileSet

Usage

```
snowprofileSet(x = list())
```

Arguments

x list of [snowprofile](#) objects

Value

a snowprofileSet

See Also

[snowprofile](#), [summary.snowprofileSet](#)

snowprofileSno	<i>Construct snowprofile object from SNO file</i>
----------------	---

Description

Read .sno files from SNOWPACK model input/output

Usage

```
snowprofileSno(Filename)
```

Arguments

Filename path to sno file

Details

Several SNOWPACK model output formats exist see [SNOWPACK documentation](#)

Definitions of SNO files are provided at <https://models.slf.ch/docserver/snowpack/html/smet.html>

Value

a [snowprofile](#) object

Author(s)

shorton

See Also[snowprofilePro](#), [snowprofilePrf](#), [snowprofileCsv](#)**Examples**

```
## Path to example prf file
Filename <- system.file('extdata', 'example.sno', package = 'sarp.snowprofile')

## Read snowprofile object
Profile <- snowprofileSno(Filename)

## Note: plot.snowprofile won't work because sno files don't have harndess

## Plot a temperautre profile
plot(snowprofileSet(list(Profile)), ColParam = 'temp')
```

snowprofileTests

*Constructor for a snowprofileTests object***Description**

Create a snowprofileTests object.

Usage

```
snowprofileTests(
  testsFrame = data.frame(type = as.character(NA), result = as.character(NA), score =
    as.double(NA), fract_char = as.character(NA), depth = as.double(NA), comment =
    as.character(NA)),
  dropNAs = TRUE
)
```

Arguments

testsFrame a data.frame listing snowpack stability tests. Rows correspond to individual tests and columns describe at least the fields c("type", "result", "fract_char", "score", "depth").

- Test **type** and **result** yield the standard 'data code' for reporting snowpack tests according to the OGRS (see Details). Following type and result combinations are allowed:
 - STV, STE, STM, STH, STN, and mixed forms STE-M, STM-H
 - CTV, CTE, CTM, CTH, CTN, and mixed forms CTE-M, CTM-H

- DTV, DTE, DTM, DTH, DTN, and mixed forms DTE-M, DTM-H
 - ECTPV, ECTP, ECTN, ECTX
 - RB, PST, DT tests are currently not supported.
 - **score**: numeric, number of taps (for CT, ECT)
 - **fract_char** corresponds to the fracture character, e.g., SP, SC, PC, RP, BRK, ...
 - **depth**: vertical location of corresponding snowpack layer (from surface)
 - potential test comment column
- dropNAs Should empty, non-mandatory columns be dropped from the final snowprofileTests object?

Details

For more information, see Canadian Avalanche Association. (2016). Observation Guidelines and Recording Standards for Weather, Snowpack, and Avalanches (OGRS). Revelstoke, BC, Canada.

Value

snowprofileTests object

Author(s)

fherla

See Also

[snowprofile](#), [snowprofileLayers](#), [snowprofileInstabilitySigns](#)

Examples

```
## create a data.frame with test observations
(testsFrame <- data.frame(type = c("CT", "ST", "ECT"),
  result = c("E-M", "M", "P"),
  score = c(10, NA, 12),
  fract_char = c("SP", NA, NA),
  depth = c(40, 40, 40),
  comment = c("some comment on first test", "", "")))

## create snowprofileTests object
tests <- snowprofileTests(testsFrame)

## create snowprofile object containing test results and check resulting object:
snowprofile(tests = tests)
```

SPgroup

Example group of snowprofiles from a mountain drainage

Description

A list of 12 snowprofile objects.

Usage

SPgroup

Format

A list with 12 entries, that are of class [snowprofile](#)

See Also

[SPpairs](#), [SPtimeline](#), [plot.snowprofileSet](#)

Examples

```
plot(SPgroup, SortMethod = 'unsorted', xticklabels = "originalIndices")
plot(SPgroup, SortMethod = 'hs', xticklabels = "originalIndices")
```

SPmalformatted

Malformatted example profiles

Description

A list with two entries, each containing a snowprofile object. Both are malformed, check out the examples in [validate_snowprofile](#) and [reformat_snowprofile](#) to learn how to fix it.

Usage

SPmalformatted

Format

A list with several entries, that are of class [snowprofile](#)

See Also

[validate_snowprofile](#), [reformat_snowprofile](#), [SPpairs](#), [SPgroup](#), [SPtimeline](#)

SPpairs	<i>Pairs of example snowprofiles</i>
---------	--------------------------------------

Description

A list with several entries, each containing a snowprofile object. Pairs of similar profiles are grouped by their names.

Usage

```
SPpairs
```

Format

A list with several entries, that are of class [snowprofile](#)

See Also

[SPgroup](#), [SPTimeline](#)

Examples

```
## Each name refers to one snowprofile:
names(SPpairs)

opar <- par(no.readonly = TRUE)
par(mfrow = c(1, 2))
plot(SPpairs$A_manual, main = 'SPpairs$A_manual')
plot(SPpairs$A_modeled, main = 'SPpairs$A_modeled')
par(opar)
```

SPTimeline	<i>Timeseries of snowprofiles #'</i>
------------	--------------------------------------

Description

Timeseries of snowprofiles #'

Usage

```
SPTimeline
```

Format

A list with several entries, that are of class [snowprofile](#)

See Also

[SPgroup](#), [SPpairs](#)

Examples

```
summary(SPtimeline)
plot(SPtimeline)
```

summary.snowprofile *Summary of a single snowprofile*

Description

Summary of a single snowprofile

Usage

```
## S3 method for class 'snowprofile'
summary(object, fast = FALSE, ...)
```

Arguments

object	snowprofile object
fast	boolean switch for twice as fast computation. downside: keep only length-1 meta data, i.e., discard latlon, or nlayers..
...	additional arguments for generic method

Details

Creates a one row data.frame where each column contains metadata.

Metadata is determines as elements of the snowprofile object list that are length = 1. An exception is made for latlon where separate columns for lat and lon are produces.

A derived value nLayers is derived by counting the number of rows in \$layers.

Value

data.frame

Author(s)

shorton

See Also

[summary.snowprofileSet](#)

Examples

```
Profile <- SPgroup[[1]]
names(Profile)
summary(Profile)
lapply(SPgroup, summary)
```

```
summary.snowprofileSet
```

Summarize multiple snowprofiles

Description

Wrapper for [summary.snowprofile](#), which only returns metadata for a single snowprofile object. `summary.snowprofileSet` provides metadata for multiple snowprofiles, which is useful for subsetting.

Usage

```
## S3 method for class 'snowprofileSet'
summary(object, ...)
```

Arguments

<code>object</code>	list of snowprofile objects
<code>...</code>	additional arguments for generic method

Value

data.frame

Author(s)

shorton

See Also

[summary.snowprofile](#), [rbind.snowprofileSet](#)

Examples

```
## Extract metadata for a group of profiles
Metadata <- summary(SPgroup)
head(Metadata)

## Subsetting profiles with Metadata
```

```
Alpine <- SPgroup[Metadata$elev > 2000]
summary(Alpine)
Shallow <- SPgroup[Metadata$hs < 150]
summary(Shallow)
Week2 <- SPTimeline[summary(SPTimeline)$date > '2017-12-15']
```

swisscode	<i>Numerical, Swiss Grain Type Code</i>
-----------	---

Description

A character array of grain types that can be translated into a numerical code by their indices.

Usage

```
swisscode
```

Format

A character array

Examples

```
print(swisscode)

## see numerical code for each grain type:
rbind(swisscode, seq(length(swisscode)))
```

validate_snowprofile	<i>Validate correctness of snowprofile object</i>
----------------------	---

Description

Validator function that checks if snowprofile standards are being met and raises an error if mandatory fields are missing or data types are incorrect. The function raises a warning when unknown field names are encountered.

Usage

```
validate_snowprofile(object, silent = FALSE)
```

Arguments

object	a snowprofile object to be validated
silent	remain silent upon error (i.e., don't raise error, but only print it)

Value

Per default an error is raised when discovered, if `silent = TRUE` the error is only printed and the error message returned (Note: a warning is never returned but only printed!). If the function is applied to multiple objects, the function returns `NULL` for each object if no error is encountered (see examples below).

See Also

[reformat_snowprofile](#)

Examples

```
## Validate individual snowprofile and raise an error
## in case of a malformed profile:

## (1) no error
validate_snowprofile(SPgroup[[1]])

## (2) malformed profile --> error
this_throws_error <- TRUE
if (!this_throws_error) {
  validate_snowprofile(SPmalformatted[[1]])
}

## Validate a list of snowprofiles and raise an error
## when the first error is encountered:
## (i.e., stop subsequent execution)

## (1) no error
lapply(SPgroup, validate_snowprofile)

## (2) malformed profile --> error
if (!this_throws_error) {
  lapply(SPmalformatted, validate_snowprofile)
}

## Validate a list of snowprofiles and continue execution,
## so that you get a comprehensive list of errors of all profiles:
if (!this_throws_error) {
  errorlist <- lapply(SPmalformatted, validate_snowprofile, silent = TRUE)
  errorlist[sapply(errorlist, function(item) !is.null(item))] # print profiles that caused errors
}
```

validate_snowprofileLayers

Validate correctness of snowprofileLayers object

Description

Validator function that checks if class standards are being met and raises an error if not.

Usage

```
validate_snowprofileLayers(object, silent = FALSE)
```

Arguments

object	to be tested
silent	remain silent upon error (i.e., don't throw error, but only print it)

Value

Per default an error is raised when discovered, if `silent = TRUE` the error is only printed and the error message returned.

writeSmet	<i>Write a SMET file</i>
-----------	--------------------------

Description

Write data into a SMET file https://models.slf.ch/docserver/meteoio/SMET_specifications.pdf

Usage

```
writeSmet(smet, filename)
```

Arguments

smet	A data structure that resembles a smet file (i.e., list containing metadata and a data.frame, see example in readSmet)
filename	Filepath to be written

Value

Generates smet file

Author(s)

fherla, shorton

See Also

[readSmet](#), [snowprofileSno](#), [snowprofilePrf](#), [snowprofilePro](#)

Examples

```
## First read example smet file provided in package
(Wx = readSmet(system.file('extdata', 'example.smet', package = 'sarp.snowprofile'))))

## Then write Wx to a new temp file and show the file
writeSmet(Wx, filename = file.path(tempdir(), 'file.smet'))
file.show(file.path(tempdir(), 'file.smet'))

## Check whether it can be read back in
(WxNew <- readSmet(file.path(tempdir(), 'file.smet'))))
```

[.snowprofileSet *Extract method*

Description

Extract method

Usage

```
## S3 method for class 'snowprofileSet'
x[i]
```

Arguments

x object from which to extract element(s) or in which to replace element(s).
i indices specifying elements to extract or replace

Value

[snowprofileSet](#) object

Index

- * **grainDict**
 - grainDict, 18
- * **object**
 - SPgroup, 56
 - SPmalformatted, 56
 - SPpairs, 57
- * **snowprofiles**
 - SPtimeline, 57
- * **snowprofile**
 - SPgroup, 56
 - SPmalformatted, 56
 - SPpairs, 57
- * **swisscode**
 - swisscode, 60
- [.snowprofileSet, 63

- char2numAspect, 3
- char2numHHI, 4, 47, 48
- computeRTA, 5, 6, 10, 48
- computeTSA, 5, 6, 10, 48

- deriveDatetag, 7
- difftime, 10

- export.snowprofileCsv, 8

- findPWL, 9, 10, 24
- format_snowprofileLayers, 12, 46

- getColoursDensity, 12, 14–17
- getColoursGrainSize, 13, 13, 15–17
- getColoursGrainType, 13, 14, 14, 15–17, 36
- getColoursHardness, 13–15, 15, 16, 17
- getColoursLWC, 13–15, 16, 17
- getColoursSnowTemp, 13–16, 17
- grainDict, 18, 36

- hasUnobservedBasalLayer, 18
- head, 30

- importRDefaultPackages, 19

- insertUnobservedBasalLayer, 19
- is.snowprofile, 20
- is.snowprofileInstabilitySigns, 20
- is.snowprofileLayers, 21
- is.snowprofileSet, 21
- is.snowprofileTests, 22

- labelPWL, 48
- labelPWL (findPWL), 9

- new_snowprofile, 22, 38
- numberOfPWLsPerVerticalLevel, 24

- plot.snowprofile, 25, 29, 40
- plot.snowprofileSet, 26, 27, 56
- print.default, 30
- print.snowprofile, 30

- rbind.snowprofile, 31, 32
- rbind.snowprofileSet, 31, 32, 59
- readSmet, 33, 62
- reformat_snowprofile, 33, 56, 61

- scanProfileDates, 34, 51, 52
- sd_sample_uncorrected, 35
- setColoursGrainType, 15, 18, 36
- simplifyGtypes, 37
- snowprofile, 5–8, 10, 18, 20, 22, 24, 25, 30, 31, 34, 38, 42, 46, 48, 53, 55–57, 60
- snowprofileCaaml, 40
- snowprofileCsv, 8, 41, 41, 54
- snowprofileCsv_advanced, 43, 43
- snowprofileInstabilitySigns, 20, 23, 39, 40, 44, 45, 55
- snowprofileLayers, 7, 10, 18, 19, 23, 24, 39, 40, 42, 44, 46, 46, 55
- snowprofilePrf, 33, 35, 50, 52, 54, 62
- snowprofilePro, 33, 35, 51, 51, 54, 62
- snowprofileSet, 21, 27, 32, 53, 63
- snowprofileSno, 33, 51, 52, 53, 62
- snowprofileTests, 22, 23, 39, 40, 44, 46, 54

SPgroup, [29](#), [56](#), [56](#), [57](#), [58](#)
SPmalformatted, [56](#)
SPpairs, [40](#), [56](#), [57](#), [58](#)
SPtimeline, [56](#), [57](#), [57](#)
summary.snowprofile, [31](#), [32](#), [40](#), [58](#), [59](#)
summary.snowprofileSet, [53](#), [58](#), [59](#)
swisscode, [60](#)

validate_snowprofile, [33](#), [38](#), [39](#), [41](#), [56](#), [60](#)
validate_snowprofileLayers, [12](#), [46](#), [48](#),
[61](#)

writeSmet, [33](#), [62](#)