Package 'sarp.snowprofile'

July 25, 2020

Version 1.0.0

Date 2020-07-20

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>).

URL http://www.avalancheresearch.ca

License CC BY-SA 4.0

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Language en-CA

Imports data.table, methods, xml2,

Depends R (>= 2.10)

Suggests knitr, rmarkdown

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 2020-07-25 11:00:02 UTC

R topics documented:

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')
}
```

`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)
```

`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,
  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
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 tuplets 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 tuplets 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$layers$g 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 tuplets 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 tuplets 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 *A data.frame storing the grain type colours*

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')
```

importRDefaultPackages *Import R_DEFAULT_PACKAGES*

Description

Import R_DEFAULT_PACKAGES

Usage

```
importRDefaultPackages()
```

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.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

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(),
  type = character(),
  band = character(),
  zone = character(),
  layers = snowprofileLayers()
)
```

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')` defaults to `'1999-12-31 UTC'`
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.
type	character string, must be either 'manual', 'vstation' or 'aggregate'
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)
layers	snowprofileLayers object

Value

snowprofile object

---

new_snowprofileLayers *Low-level constructor for a snowprofileLayers object*

Description

Low-cost, efficient constructor function to be used by users who know what they're doing.

Important: Make sure the last row of the data.frame corresponds to the snow surface. No checks incorporated for this low-level constructor!

Usage

```
new_snowprofileLayers(...)
```

Arguments

...	see snowprofileLayers

Value

snowprofileLayers object as data.frame with strings as factors

`plot.snowprofile` *Plot hardness profile*

Description

Plot hardness profile

Usage

```
## S3 method for class 'snowprofile'
plot(
  x,
  TempProfile = TRUE,
  Col = sapply(x$layers$gtype, getColoursGrainType),
  TopDown = FALSE,
  axes = TRUE,
  xlab = "",
  emphasizeLayers = FALSE,
  emphasis = "95",
  failureLayers = FALSE,
  failureLayers.cex = 1,
  ...
)
```

Arguments

`x`	snowprofile object
`TempProfile`	draw unscaled temperature profile (default = TRUE)
`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 a red arrow
`failureLayers.cex`	factor to shrink or enlarge the arrow
`...`	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.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 = "time",
  ColParam = "gtype",
  TopDown = FALSE,
  DateStart = NA,
  DateEnd = NA,
  ylim = NULL,
  OutlineLyrs = FALSE,
  HorizGrid = TRUE,
  main = NA,
  ylab = NA,
  xlab = NA,
  box = TRUE,
  labelOriginalIndices = FALSE,
  yPadding = 10,
  xPadding = 0.5,
  ...
)
```

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).
ylim	Vertical range of plot
OutlineLyrs	Switch for outlining layers (default = FALSE)
HorizGrid	Draw horizontal grid at layer heights (default = TRUE)
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)
labelOriginalIndices	Label the original (i.e. prior to sorting) indices of the profiles at the x-axis? (default = FALSE)
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'.
...	Additional parameters passed to plot()

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')

## Additional examples of plot dimensions and labelling
## Label the indices of the profiles in the list:
plot(SPgroup, SortMethod = 'elev', labelOriginalIndices = TRUE)
##  ... and with minimized axis limits:
plot(SPgroup, SortMethod = 'elev', labelOriginalIndices = TRUE,
       yPadding = 0, xPadding = 0, xaxs = 'i', yaxs = 'i')
##  sorted by depth, and without box:
plot(SPgroup, SortMethod = 'hs', TopDown = TRUE, box = FALSE)
```

`print.snowprofile` *Print snowprofile object*

Description

Print snowprofile object

Usage

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

Arguments

`x`	snowprofile object
`pretty`	pretty print the object (data.frame-like instead of list-like)
`nLayers`	only print the first few layers (cf., head)
`...`	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 *Convert snowprofile into data.frame with columns for metadata*

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 snowprofilSet 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` *Parse a SMET file*

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

snowprofileSno, snowprofilePrf, snowprofilePro

Examples

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

reformat_snowprofile *Reformat a malformatted snowprofile object*

Description

Reformat a malformatted snowprofile object. A malformatted 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 malformatted 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 malformatted 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)
```

`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 • iacs: scale defined by the *International Classification of Seasonal Snow on the Ground* • iacs2: scale defined by the *International Classification of Seasonal Snow on the Ground* 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, https://doi.org/10.5194/nhess-20-1557-2020, 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')
```

`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, tz = "UTC"),
  latlon = as.double(c(NA, NA)),
  elev = as.double(NA),
  angle = as.double(NA),
  aspect = as.double(NA),
  hs = as.double(NA),
  type = "manual",
  band = as.character(NA),
  zone = as.character(NA),
  layers = snowprofileLayers(formatTarget = "FALSE", 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. `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.
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)
layers	snowprofileLayers 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, 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

Read a Caaml file into a snowprofile object

Usage

```
snowprofileCaaml(caamlFile, sourceType = NA)
```

Arguments

caamlFile	'path/to/file.caaml'
sourceType	choose 'manual', 'modeled', 'vstation', 'aggregate' or 'whiteboard'; if NA, the default will be chosen by snowprofile.

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')
```

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",
  ...
)
```

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|crust), or as one-digit code specifying the primary grain type *if* another column is provided that specifies crusts. See 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. • profile specific info: station, station_id, datetime, latlon, elev, angle, aspect, type (see snowprofile) • layer specific info: deposition date, grain size, ssi, ... (see snowprofileLayers)

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).

Value

snowprofile object

Author(s)

fherla

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')
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'`.

## let's assume you want to read the csv file an 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.
```

`snowprofileLayers` *High-level 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 an ascending manner, i.e. last row corresponds to snow surface. If only depth is given, the layer thickness of the lowermost layer will be set to a default value (100 cm) to be able to convert to height (i.e. important for subsequent package routines). If the columns are not of equal lengths, their values will be recycled (default data.frame mechanism), but a warning will be issued. Certain columns will be auto-filled (format_snowprofileLayers). Instead of individual layer characteristics, a data.frame can be provided, which will be converted into a snowprofileLayers class. Calls low-level constructor new_snowprofileLayers and asserts correctness through 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),
  ssi = as.double(NA),
  ...,
  hs = as.double(NA),
  formatTarget = "all",
  layerFrame = NA,
  validate = TRUE,
  dropNAs = TRUE
)
```

Arguments

height	height vector (cm)
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)
ssi	snow stability index (numeric)
...	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 or depth & thickness vector

formatTarget	string indicating which layer characteristics should be auto-filled, e.g. 'all' (default), 'height', 'depth', 'thickness', 'none'
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(dropNAs = FALSE)


## convert and recycle character hardness (i.e., warning issued):
snowprofileLayers(height = c(10, 25, 50),
                  hardness = char2numHHI('1F+'),
                  gtype = c('FC', NA, 'PP'))


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

snowprofileLayers(layerFrame = df)
```

---

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 of 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 of 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` *Constructor for class snowprofileSet*

Description

Constructor for class snowprofileSet

Usage

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

Arguments

x	list of snowprofile objects

Value

a snowprofileSet

See Also

snowprofile, summary.snowprofileSet

`snowprofileSno` *Construct snowprofile object from SNO file*

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')
```

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

Examples

```
plot(SPgroup, SortMethod = 'unsorted', labelOriginalIndices = TRUE)
```

SPmalformatted *Malformatted example profiles*

Description

A list with two entries, each containing a snowprofile object. Both are malformatted, 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 *Pairs of example snowprofiles*

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 *Timeseries of snowprofiles #'*

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, ...)
```

Arguments

`object`	snowprofile object
`...`	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

`object`	list of snowprofile objects
`...`	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 *Numerical, Swiss Grain Type Code*

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 *Validate correctness of snowprofile object*

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 malformatted profile:

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

## (2) malformatted 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) malformatted 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)
}
```

`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.

[.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