Package 'swfscAirDAS'

August 9, 2023

Title Southwest Fisheries Science Center Aerial DAS Data Processing

Version 0.3.0

Description Process and summarize aerial survey 'DAS' data (AirDAS)
<https://swfsc-publications.fisheries.noaa.gov/publications/TM/SWFSC/NOAA-TM-NMFS-SWFSC-185.PDF>
collected using an aerial survey program from the
Southwest Fisheries Science Center (SWFSC)
<https://www.fisheries.noaa.gov/west-coast/science-data/california-current-marine-mammal-assessment-program>.
PDF files detailing the relevant AirDAS data formats are included in this package.

URL https://smwoodman.github.io/swfscAirDAS/,
https://github.com/smwoodman/swfscAirDAS/

BugReports https://github.com/smwoodman/swfscAirDAS/issues/

Depends R (>= 4.0.0)

Imports dplyr, lubridate, magrittr, methods, parallel, purrr, readr, rlang, stringr, swfscDAS (>= 0.3.0), swfscMisc, tidyr

Suggests knitr, rmarkdown, testthat (>= 2.1.0), tibble

License CC0

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Sam Woodman [aut, cre] (<https://orcid.org/0000-0001-6071-8186>)

Maintainer Sam Woodman <sam.woodman@noaa.gov>

Repository CRAN

Date/Publication 2023-08-08 23:50:02 UTC

R topics documented:

`swfscAirDAS-package` *Southwest Fisheries Science Center Aerial Survey DAS*

Description

Process and summarize aerial survey DAS data

Details

This package contains functions designed for processing and analyzing aerial survey DAS data (AirDAS) collected using one of the following Southwest Fisheries Science Center (SWFSC) programs: PHOCOENA, SURVEY, CARETTA, or TURTLE (such as TURTLEP or TURTLE 4D). Functionality includes checking AirDAS data for data entry errors, reading AirDAS data into a data frame, processing this data (extracting state and condition information for each AirDAS event), and summarizing sighting and effort information.

Author(s)

Sam Woodman <sam.woodman@noaa.gov>

See Also

https://smwoodman.github.io/swfscAirDAS/

`airdas_check` *Check AirDAS file*

Description

Check that AirDAS file has accepted formatting and values

Usage

```
airdas_check(
  file,
  file.type = c("turtle", "caretta", "phocoena"),
  skip = 0,
  file.out = NULL,
  sp.codes = NULL,
  print.transect = TRUE
)
```

Arguments

`file`	filename(s) of one or more AirDAS files
`file.type`	character; indicates the program used to create `file`. Must be one of: "turtle", "caretta", "survey", or "phocoena" (case sensitive). Default is "turtle". Passed to `airdas_read`
`skip`	integer: see `read_fwf`. Default is 0. Passed to `airdas_read`
`file.out`	character; filename to which to write the error log. Should be a text or CSV file. Default is `NULL`
`sp.codes`	character; filename of .dat file from which to read accepted species codes. If `NULL`, default (internal) file will be used. Default is `NULL`
`print.transect`	logical; indicates if a table with all the transect numbers in the `x` should be printed using `table`. Default is `TRUE`

Details

The default (internal) `sp.codes` file is located at `system.file("SpCodesAirDAS.dat", package = "swfscAirDAS")`.

To see the checks performed by this function, you can access the PDF locally at `system.file("AirDAS_check.pdf", package = "swfscAirDAS")`, or online at https://github.com/smwoodman/swfscAirDAS/blob/master/inst/AirDAS_check.pdf

Checks that are not done by this function that may be of interest:

- Check for valid fish ball/mola/jelly/crab pot codes
- Check that datetimes are sequential, meaning they 1) are the same as or 2) come after the previous event

Value

A data frame with five columns that list information about errors found in the AirDAS files: the file name, line number, index (row number) from the `airdas_read(file)` data frame, 'ID' (pre-Data# columns from the DAS file), and description of the issue. This data frame is sorted by the 'Description' column. If there are multiple issues with the same line, the issue descriptions are concatenated together using `paste(..., collapse = "; ")`

If `print.transect` is `TRUE`, then the output of `table(x$Data1[x$Event == "T"], useNA = "always")`, where `x` is the output of `airdas_read(file, ...)` is printed

If `file.out` is not `NULL`, then the error log is also written to the file (e.g., a .txt or .csv file) specified by `file.out`

See Also

https://smwoodman.github.io/swfscAirDAS/

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
if (interactive()) airdas_check(y, print.transect = TRUE)
```

---

airdas_chop_condition *Chop AirDAS data - condition*

Description

Chop AirDAS data into a new effort segment every time a condition changes

Usage

```
airdas_chop_condition(x, ...)

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

## S3 method for class 'airdas_df'
airdas_chop_condition(
  x,
  conditions,
  seg.min.km = 0.1,
  distance.method = NULL,
  num.cores = NULL,
  ...
)
```

Arguments

x	airdas_df object, or a data frame that can be coerced to a airdas_df object. This data must be filtered for 'OnEffort' events; see the Details section below
...	ignored
conditions	the conditions that trigger a new segment; see airdas_effort
seg.min.km	numeric; minimum allowable segment length (in kilometers). Default is 0.1. See the Details section below for more information
distance.method	character; see airdas_effort. Default is NULL since these distances should have already been calculated in airdas_effort
num.cores	See airdas_effort

Details

WARNING - do not call this function directly! It is exported for documentation purposes, but is intended for internal package use only.

This function is intended to only be called by airdas_effort when the "condition" method is specified. Thus, x must be filtered for events (rows) where either the 'OnEffort' column is TRUE or the 'Event' column is either "E" or "O"; see airdas_effort for more details. This function chops each continuous effort section (henceforth 'effort sections') in x into modeling segments (henceforth 'segments') by creating a new segment every time a condition changes. Each effort section runs from a T/R event to its corresponding E/O event. After chopping, airdas_segdata is called (with segdata.method = "maxdist") to get relevant segdata information for each segment.

Changes in the one of the conditions specified in the conditions argument triggers a new segment. An exception is when multiple condition changes happen at the same location, such as a 'TVPAW' series of events. When this happens, no segments of length zero are created; rather, a single segment is created that includes all of the condition changes (i.e. all of the events in the event series) that happened during the series of events (i.e. at the same location). Note that this combining of events at the same Lat/Lon happens even if seg.min.km = 0.

In addition, (almost) all segments whose length is less than seg.min.km are combined with the segment immediately following them to ensure that the length of (almost) all segments is at least seg.min.km. This allows users to account for situations where multiple conditions, such as Beaufort and a viewing condition, change in rapid succession, say <0.1 km apart. When segments are combined, a message is printed, and the condition that was recorded for the maximum distance within the new segment is reported. See airdas_segdata, segdata.method = "maxdist", for more details about how the segdata information is determined. The only exception to this rule is if the short segment ends in an "E" or an "O" event, meaning it is the last segment of the effort section. Since in this case there is no 'next' segment, this short segment is left as-is.

If the column dist_from_prev does not exist, the distance between subsequent events is calculated as described in airdas_effort

Value

List of two data frames:

- x, with columns added for the corresponding unique segment code and number

- segdata: data frame with one row for each segment, and columns with relevant data (see airdas_effort for specifics)

`airdas_chop_equallength` *Chop AirDAS data - equal length*

Description

Chop AirDAS data into equal-length effort segments, averaging conditions by segment

Usage

```
airdas_chop_equallength(x, ...)

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

## S3 method for class 'airdas_df'
airdas_chop_equallength(
  x,
  conditions,
  seg.km,
  randpicks.load = NULL,
  distance.method = NULL,
  num.cores = NULL,
  ...
)
```

Arguments

x	airdas_df object, or a data frame that can be coerced to a airdas_df object. This data must be filtered for 'OnEffort' events; see the Details section below
...	ignored
conditions	see airdas_effort
seg.km	numeric; target segment length in kilometers
randpicks.load	character, data frame, or NULL. If character, must be filename of past randpicks output to load and use (passed to file argument of read.csv). If data frame, randpicks values will be extracted from the data frame. If NULL, new randpicks values will be generated by the function
distance.method	character; see airdas_effort. Default is NULL since these distances should have already been calculated in airdas_effort
num.cores	See airdas_effort

Details

WARNING - do not call this function directly! It is exported for documentation purposes, but is intended for internal package use only.

This function is intended to only be called by `airdas_effort` when the "equallength" method is specified. Thus, `x` must be filtered for events (rows) where either the 'OnEffort' column is `TRUE` or the 'Event' column is either "E" or "O"; see `airdas_effort` for more details. This function chops each continuous effort section (henceforth 'effort sections') in `x` into modeling segments (henceforth 'segments') of equal length. Each effort section runs from a "T"/"R" event to its corresponding "E"/"O" event. After chopping, `airdas_segdata` is called to get relevant segdata information for each segment.

When chopping the effort sections in segments of length `seg.km`, there are several possible scenarios:

- The extra length remaining after chopping is greater than or equal to half of the target segment length (i.e. `>= 0.5*seg.km`): the extra length is assigned to a random portion of the effort section as its own segment (see Fig. 1a)
- The extra length remaining after chopping is less than half of the target segment length (i.e. `< 0.5*seg.km`): the extra length is added to one of the (randomly selected) equal-length segments (see Fig. 1b)
- The length of the effort section is less than or equal to the target segment length: the entire segment becomes a segment (see Fig. 1c)
- The length of the effort section is zero: a segment of length zero. If there are more than two events (the "T"/R" and "E"/"O" events), the function throws a warning

Therefore, the length of each segment is constrained to be between one half and one and one half of `seg.km` (i.e. `0.5*seg.km` <= segment length `>=1.5*seg.km`), and the central tendency is approximately equal to the target segment length. The only exception is when a continuous effort section is less than one half of the target segment length (i.e. `< 0.5*seg.km`; see Fig. 1c).

Note the PDF with Figs. 1a - 1c is included in the package, and can be found at: `system.file("AirDAS_chop_equallength_ package = "swfscAirDAS")`

'Randpicks' is a record of the random assignments that were made when chopping the effort sections into segments, and can be saved to allow users to recreate the same random allocation of extra km when chopping. The randpicks returned by this function is a data frame with two columns: the number of the effort section and the randpick value. Users should save the randpicks output to a CSV file, which then can be specified using the `randpicks.load` argument to recreate the same effort segments from `x` (i.e., using the same AirDAS data) in the future. Note that when saving with `write.csv`, users must specify `row.names = FALSE` so that the CSV file only has two columns. For an example randpicks file, see `system.file("airdas_sample_randpicks.csv", package = "swfscAirDAS")`

If the column `dist_from_prev` does not exist, the distance between subsequent events is calculated as described in `airdas_effort`

Value

List of three data frames:

- `x`, with columns added for the corresponding unique segment code and number

- segdata: data frame with one row for each segment, and columns with relevant data (see airdas_effort for specifics)
- randpicks: data frame with record of length allocations (see Details section above)

`airdas_chop_section` *Chop AirDAS data - section*

Description

Chop AirDAS data into effort segments by continuous effort section

Usage

```
airdas_chop_section(x, ...)

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

## S3 method for class 'airdas_df'
airdas_chop_section(
  x,
  conditions,
  distance.method = NULL,
  num.cores = NULL,
  ...
)
```

Arguments

`x`	`airdas_df` object, or a data frame that can be coerced to a `airdas_df` object. This data must be filtered for 'OnEffort' events; see the Details section below
`...`	ignored
`conditions`	see `airdas_effort`
`distance.method`	character; see `airdas_effort`. Default is `NULL` since these distances should have already been calculated in `airdas_effort`
`num.cores`	See `airdas_effort`

Details

WARNING - do not call this function directly! It is exported for documentation purposes, but is intended for internal package use only.

This function is simply a wrapper for `airdas_chop_equallength`. It calls `airdas_chop_equallength`, with `seg.km` set to a value larger than the longest continuous effort section in `x`. Thus, the effort is 'chopped' into the continuous effort sections and then summarized.

See the Examples section for an example where the two methods give the same output. Note that the longest continuous effort section in the sample data is ~32km.

For an example of how to summarize data by transect, see `vignette("swfscAirDAS")`. In short, if looking to group by individual transects, use `segdata$transect_idx <- cumsum(segdata$event == "T")` to create a column with a transect index. Then you can use `group_by(transect_idx)` and `summarise` to summarise the desired data by transect

Value

See `airdas_chop_equallength`. The randpicks values will all be `NA`

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.proc <- airdas_process(y)

y.eff1 <- airdas_effort(y.proc, method = "equallength", seg.km = 35, num.cores = 1)
y.eff2 <- airdas_effort(y.proc, method = "section", num.cores = 1)

all.equal(y.eff1, y.eff2)
```

`airdas_comments` *Extract comments from AirDAS data*

Description

Extract comments from `airdas_dfr` or `airdas_df` object

Usage

```
airdas_comments(x)

## S3 method for class 'data.frame'
airdas_comments(x)

## S3 method for class 'airdas_df'
airdas_comments(x)

## S3 method for class 'airdas_dfr'
airdas_comments(x)
```

Arguments

`x`	`airdas_dfr` or `airdas_df` object, or a data frame that can be coerced to a `airdas_dfr` object

Details

This function recreates the comment strings by pasting the Data# columns back together for the C events (comments)

See the examples section for how to search for comments with the phrase "record" to determine what extra information (e.g. molas) was being recorded vs ignored.

Value

x, filtered for C events and with the added column comment_str containing the concatenated comment strings

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.read <- airdas_read(y, file.type = "turtle")

airdas_comments(y.read)

# Extract all comments containing "record"
y.comm <- airdas_comments(y.read)
y.comm[grepl("record", y.comm$comment_str, ignore.case = TRUE), ]

# Extract all comments containing "record", but not "recorder"
y.comm <- airdas_comments(y.read)
y.comm[grepl("record", y.comm$comment_str, ignore.case = TRUE) &
         !grepl("recorder", y.comm$comment_str, ignore.case = TRUE), ]

# Join comments with processed data
dplyr::left_join(y.read, y.comm[, c("file_das", "line_num", "comment_str")],
                 by = c("file_das", "line_num"))
```

---

airdas_comments_process

Process comments in AirDAS data

Description

Extract miscellaneous information recorded in AirDAS data comments, i.e. comment-data

Usage

```
airdas_comments_process(x, ...)

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

## S3 method for class 'airdas_dfr'
```

```
airdas_comments_process(x, comment.format = NULL, ...)

## S3 method for class 'airdas_df'
airdas_comments_process(x, comment.format = NULL, ...)
```

Arguments

x	airdas_dfr or airdas_df object, or a data frame that can be coerced to a airdas_dfr object
...	ignored
comment.format	list; default is NULL. See the 'Using comment.format' section

Details

Historically, project-specific or miscellaneous data have been recorded in AirDAS comments using specific formats and character codes. This functions identifies and extracts this data from the comment text strings. However, different data types have different comment-data formats. Specifically, TURTLE and PHOCOENA comment-data uses identifier codes that each signify a certain data pattern, while other comment-data (usually that of CARETTA) uses data separated by some delimiter.

Value

x, filtered for comments with recorded data, with the following columns added:

- comment_str: the full comment string
- Misc#: Some number of descriptor columns. There should be n columns, although the minimum number will be two columns
- Value: Associated count or percentage for TURTLE/PHOCOENA data
- flag_check: logical indicating if the TURTLE/PHOCOENA comment string was longer than an expected number of characters, and thus should be manually inspected

See the additional sections for more context. If comment.format is NULL, then the output data frame would two Misc# columns: a level one descriptor, e.g. "Fish ball" or "Jellyfish", and a level two descriptor, e.g. s, m, or c. However, if comment.format$n is say 4, then the output data frame would have columns Misc1, Misc2, Misc3, and Misc4.

Messages are printed if either comment.format is not NULL and not comment-data is identified using comment.format, or if x has TURTLE/PHOCOENA data but no TURTLE/PHOCOENA comment-data

TURTLE and PHOCOENA comment-data

Current supported data types are: fish balls, molas, jellyfish, and crab pots. See any of the AirDAS format PDFs (airdas_format_pdf) for information about the specific codes and formats used to record this data. All comments are converted to lower case for processing to avoid missing data.

These different codes contain (at most): a level one descriptor (e.g. fish ball or crab pot), a level two descriptor (e.g. size or jellyfish species), and a value (a count or percentage). Thus, the extracted data are returned together in this structure. The output data frame is long data, i.e. it has one piece

of information per line. For instance, if the comment is "fb1s fb1m", then the output data frame will have one line for the small fish ball and one for the medium fish ball. See Value section for more details.

Currently this function only recognizes mola data recorded using the "m1", "m2", and "m3" codes (small, medium, and large mola, respectively). Thus, "mola" is not recognized and processed.

The following codes are used for the level two descriptors:

Description	*Code*
Small	s
Medium	m
Large	l
Unknown	u
Chrysaora	c
Moon jelly	m
Egg yolk	e
Other	o

Using `comment.format`

`comment.format` is a list that allows the user to specify the comment-data format. To use this argument, data must be separated by a delimiter. This list must contain three named elements:

- n: A single number indicating the number of elements of data in each comment. Must equal the length of `type`. A comment must contain exactly this number of `sep` to be recognized as comment-data
- sep: A single string indicating the field separator string (delimiter). Values within each comment are separated by this string. Currently accepted values are ";" and ","
- type: A character vector of length `n` indicating the data type of each data element (column). All values must be one of: "character", "numeric", or "integer".

For instance, for most CARETTA data `comment.format` should be `list(n = 5, sep = ";", type = c("character", "character", "numeric", "numeric", "character"))`

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.proc <- airdas_process(y)

airdas_comments_process(y.proc)
```

`airdas_df-class` `airdas_df` *class*

Description

The airdas_df class is a subclass of data.frame, created to provide a concise and robust way to ensure that the input to downstream AirDAS processing functions, such as airdas_sight, adheres to certain requirements. Specifically, objects of class airdas_df are data frames with specific column names and classes, as detailed in the 'Properties of airdas_df' section. In addition, airdas_df objects have no NA values in the 'Lat' 'Lon', or 'DateTime' columns. Objects of class airdas_df are created by airdas_process or as_airdas_df, and are intended to be passed directly to DAS processing functions such as airdas_sight.

Subsetting, say for a specific date or transect number, or otherwise altering an object of class airdas_df will cause the object to drop its airdas_df class attribute, although note that combining two airdas_df objects using rbind will return an object with a airdas_df class attribute. If this object is then passed to a DAS processing function such as airdas_sight, the function will try to coerce the object to a airdas_df object.

Properties of airdas_df objects

Objects of class airdas_df have a class attribute of c("airdas_df", "data.frame"). All values in the OnEffort column must be TRUE or FALSE (no NA values). All on effort events must have non-NA Lat/Lon/DateTime values, and there must be no events with a "#" event code (deleted event). Like airdas_dfr events, there must be a file_type column where all values are one of: "turtle", "caretta", "survey", or "phocoena" (case sensitive; see airdas_read for more details about file types).

In addition, airdas_df objects must have the following column names and classes:

Column name	*Column class*
Event	"character"
DateTime	c("POSIXct", "POSIXt")
Lat	"numeric"
Lon	"numeric"
OnEffort	"logical"
Trans	"character"
Bft	"numeric"
CCover	"numeric"
Jelly	"numeric"
HorizSun	"numeric"
VertSun	"numeric"
HKR	"character"
Haze	"logical"
Kelp	"logical"
Red tide	"logical"
AltFt	"numeric"
SpKnot	"numeric"
ObsL	"character"
ObsB	"character"
ObsR	"character"
Rec	"character"
VLI	"character"
VLO	"character"

VB	"character"
VRI	"character"
VRO	"character"
Data1	"character"
Data2	"character"
Data3	"character"
Data4	"character"
Data5	"character"
Data6	"character"
Data7	"character"
EffortDot	"logical"
EventNum	"character"
file_das	"character"
line_num	"integer"
file_type	"character"

See Also

as_airdas_df

`airdas_dfr-class` `airdas_dfr` *class*

Description

The `airdas_dfr` class is a subclass of `data.frame`, created to provide a concise and robust way to ensure that the input to `airdas_process`adheres to certain requirements. Specifically, objects of class `airdas_dfr` are data frames with specific column names and classes, as detailed in the 'Properties of `airdas_dfr`' section. Objects of class `airdas_dfr` are created by `airdas_read` or `as_airdas_dfr`, and are intended to be passed directly to `airdas_process`.

Subsetting or otherwise altering an object of class `airdas_dfr` will cause the object to drop its `airdas_dfr` class attribute, although note that combining two `airdas_dfr` objects using `rbind` will return an object with a `airdas_dfr` class attribute. `airdas_process` will then try to coerce the object to a `airdas_dfr` object. It is **strongly** recommended to pass an object of class `airdas_dfr` to `airdas_process` before subsetting, e.g. for events from a certain date range.

Properties of `airdas_dfr` objects

Objects of class `airdas_dfr` have a class attribute of `c("airdas_dfr", "data.frame")`. They must have a column `file_type` where all values are one of: "turtle", "caretta", "survey", or "phocoena" (case sensitive; see `airdas_read` for more details). `airdas_dfr` objects also must not have any `NA` event codes.

In addition, they must have the following column names and classes:

Column name	*Column class*
Event	"character"

EffortDot	"logical"
DateTime	c("POSIXct", "POSIXt")
Lat	"numeric"
Lon	"numeric"
Data1	"character"
Data2	"character"
Data3	"character"
Data4	"character"
Data5	"character"
Data6	"character"
Data7	"character"
EventNum	"character"
file_das	"character"
line_num	"integer"
file_type	"character"

See Also

as_airdas_dfr

`airdas_effort` *Summarize AirDAS effort*

Description

Chop AirDAS data into effort segments

Usage

```
airdas_effort(x, ...)

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

## S3 method for class 'airdas_df'
airdas_effort(
  x,
  method = c("condition", "equallength", "section"),
  conditions = NULL,
  distance.method = c("greatcircle", "lawofcosines", "haversine", "vincenty"),
  num.cores = NULL,
  angle.min = 12,
  bft.max = 5,
  ...
)
```

Arguments

x	airdas_df object; output from airdas_process, or a data frame that can be coerced to a airdas_df object
...	arguments passed to the chopping function specified using method, such as seg.km or seg.min.km
method	character; method to use to chop AirDAS data into effort segments Can be "condition", "equallength", "section", or any partial match thereof (case sensitive)
conditions	character vector of names of conditions to include in segdata output. These values must be column names from the output of airdas_process, e.g. 'Bft', 'CCover', etc. The default is NULL, in which case all relevant conditions will be included. If method == "condition", then these also are the conditions which trigger segment chopping when they change.
distance.method	character; method to use to calculate distance between lat/lon coordinates. Can be "greatcircle", "lawofcosines", "haversine", "vincenty", or any partial match thereof (case sensitive). Default is "greatcircle"
num.cores	Number of CPUs to over which to distribute computations. Defaults to NULL, which uses one fewer than the number of cores reported by detectCores Using 1 core likely will be faster for smaller datasets
angle.min	passed to airdas_sight
bft.max	numeric; the maximum Beaufort (column 'Bft') for which to mark a sighting as TRUE in 'included' (see Details). Default is 5.

Details

This is the top-level function for chopping processed AirDAS data into modeling segments (henceforth 'segments'), and assigning sightings and related information (e.g., weather conditions) to each segment. This function returns data frames with all relevant information for the effort segments and associated sightings ('segdata' and 'sightinfo', respectively). Before chopping, the AirDAS data is filtered for events (rows) where either the 'OnEffort' column is TRUE or the 'Event' column is "E" or "O". In other words, the data is filtered for continuous effort sections (henceforth 'effort sections'), where effort sections run from "T"/"R" to "E"/"O" events (inclusive), and then passed to the chopping function specified using method. All on effort events must not have NA Lat or Lon values; note Lat/Lon values for 1 events were 'filled in' in airdas_process.

The following chopping methods are currently available: "condition", "equallength", and "section". When using the "condition" method, effort sections are chopped into segments every time a condition specified in conditions changes, thereby ensuring that the conditions are consistent across the entire segment. See airdas_chop_condition for more details about this method, including arguments that must be passed to it via ....

The "equallength" method consists of chopping effort sections into equal-length segments of length seg.km, and doing a weighted average of the conditions for the length of that segment. See airdas_chop_equallength for more details about this method, including arguments that must be passed to it via ....

The "section" method involves 'chopping' the effort into continuous effort sections, i.e. each continuous effort section is a single effort segment. See airdas_chop_section for more details about this method.

The distance between the lat/lon points of subsequent events is calculated using the method specified in `distance.method`. If "greatcircle", `distance_greatcircle` is used, while `distance` is used otherwise. See `airdas_sight` for how the sightings are processed.

The sightinfo data frame includes the column 'included', which is used in `airdas_effort_sight` when summarizing the number of sightings and animals for selected species. `airdas_effort_sight` is a separate function to allow users to personalize the 'included' values as desired for their specific analysis. By default, i.e. in the output of this function, 'included' is `TRUE` if: the sighting was a standard sighting (see `airdas_sight`) and in a Beaufort sea state less than or equal to 'btf.max'.

Value

List of three data frames:

- segdata: one row for every segment, and columns for information including unique segment number, event code that started the associated continuous effort section, the starting and ending line of the segment in the DAS file (stlin, endlin), start/end/midpoint coordinates(lat1/lon1, lat2/lon2, and mlat/mlon, respectively), the start/end/midpoint date/time of the segment (DateTime1, DateTime2, and mDateTime, respectively; mDateTime is the average of DateTime1 and DateTime2), segment length (dist), and conditions (e.g. Beaufort)
- sightinfo: details for all sightings in `x`, including: the unique segment number it is associated with, segment mid points (lat/lon), the 'included' column described in the Details section, and the output information described in `airdas_sight`
- randpicks: see `airdas_chop_equallength`. `NULL` if using "condition" method.

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.proc <- airdas_process(y)

airdas_effort(
  y.proc, method = "condition", conditions = "Bft", seg.min.km = 0.05,
  num.cores = 1
)

y.rand <- system.file("airdas_sample_randpicks.csv", package = "swfscAirDAS")
airdas_effort(
  y.proc, method = "equallength", conditions = c("Bft", "CCover"),
  seg.km = 3, randpicks.load = y.rand, num.cores = 1
)

airdas_effort(y.proc, method = "section", num.cores = 1)
```

---

airdas_effort_sight *Summarize AirDAS sightings by effort segment*

Description

Summarize number of sightings and animals for selected species by segment

Usage

```
airdas_effort_sight(x.list, sp.codes, sp.events = c("S", "t"))
```

Arguments

x.list	list; output of airdas_effort
sp.codes	character; species code(s) to include in segdata. These code(s) will be converted to lower case to match airdas_sight
sp.events	character; event code(s) to include in the sightinfo output. This argument supersedes the 'included' value when determining whether a sighting is included in the segment summaries. Must be one or more of: "S", "t" (case-sensitive). The default is that all of these event codes are kept

Details

This function takes the output of airdas_effort and adds columns for the number of sightings (nSI) and number of animals (ANI) for selected species (selected via sp.codes) for each segment to the segdata element of x.list. However, only sightings with an included value of TRUE (included is a column in sightinfo) are included in the summaries. Having this step separate from airdas_effort allows users to personalize the included values as desired for their analysis.

Value

A list, identical to x.list except for 1) the nSI and ANI columns added to x.list$segdata, one each for each element of sp.codes, and 2) the included column of x.list$sightinfo, which has been set as FALSE for sightings of species not listed in sp.codes

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.proc <- airdas_process(y)
y.cond <- airdas_effort(
  y.proc, method = "condition", conditions = "Bft", seg.min.km = 0.05,
  num.cores = 1
)

airdas_effort_sight(y.cond, sp.codes = c("mn", "bm"))
```

---

airdas_format_pdf *Aerial DAS format requirements*

Description

Access and save local PDF documents describing the data format of the different file types supported by swfscAirDAS

Usage

```
airdas_format_pdf(file, file.type = c("phocoena", "caretta", "turtle"), ...)
```

Arguments

file	character; the name of the file where the PDF will be saved
file.type	character; indicates which data format PDF to extract. Must be one of: "turtle", "caretta", "survey", or "phocoena" (case sensitive)
...	passed to file.copy; might included named argument overwrite

Details

This function is a wrapper function for file.copy. It saves a PDF document describing the specified aerial DAS data format requirements by copying the PDF document to file

The PDF files can also be manually copied or downloaded from:

PHOCOENA

- Can be copied from: system.file("AirDAS_Format_PHOCOENA.pdf", package = "swfscAirDAS")
- Can be downloaded from: https://github.com/smwoodman/swfscAirDAS/blob/master/inst/AirDAS_Format_PHOCOENA.pdf

CARETTA

- Can be copied from: system.file("AirDAS_Format_CARETTA.pdf", package = "swfscAirDAS")
- Can be downloaded from: https://github.com/smwoodman/swfscAirDAS/blob/master/inst/AirDAS_Format_CARETTA.pdf

TURTLE

- Can be copied from: system.file("AirDAS_Format_TURTLE.pdf", package = "swfscAirDAS")
- Can be downloaded from: https://github.com/smwoodman/swfscAirDAS/blob/master/inst/AirDAS_Format_TURTLE.pdf

Value

output of file.copy: TRUE if writing of file was successful, and FALSE otherwise

See Also

https://smwoodman.github.io/swfscAirDAS/

Examples

```
if (interactive()) {
  airdas_format_pdf(
    "AirDAS_Format_TURTLE.pdf", file.type  = "turtle",
    overwrite = FALSE
  )
}
```

`airdas_process` *Process aerial survey DAS data*

Description

Process AirDAS data (the output of `airdas_read`), including extracting state and condition information for each AirDAS event

Usage

```
airdas_process(x, ...)

## S3 method for class 'character'
airdas_process(x, ...)

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

## S3 method for class 'airdas_dfr'
airdas_process(
  x,
  days.gap.part = 0.5/24,
  days.gap.full = 12/24,
  gap.message = FALSE,
  reset.transect = TRUE,
  trans.upper = FALSE,
  ...
)
```

Arguments

`x`	an object of class `airdas_dfr` object, an object that can be coerced to class `airdas_dfr`, or a character (filepath) which is first passed to `airdas_read`
`...`	passed to `airdas_read` if `x` is a character. Otherwise ignored
`days.gap.part`	numeric of length 1; time gap (in days) used to identify when a 'partial reset' is performed, i.e. when propagated info (weather, observers, etc) is reset. Default is 30 minutes; must be less than or equal to `days.gap.full`
`days.gap.full`	numeric of length 1; time gap (in days) used to identify when a 'full reset; is performed, i.e. when all info (transect number and propagated info) is reset. Default is 12 hours; must be greater than `days.gap.part`
`gap.message`	logical; default is `FALSE`. Indicates if messages should be printed detailing which row(s) of the output data frame were partially or fully reset
`reset.transect`	logical; default is `TRUE`. Indicates if propagated info (weather, observers, etc) should be reset to `NA` when beginning a new transect. See Details section
`trans.upper`	logical; indicates if all transect codes should be capitalized using `toupper`. Default is `FALSE`

Details

If x is a character, it is assumed to be a filepath and first passed to airdas_read. This output is then processed.

This function cannot handle concatenated airdas_dfr objects of multiple file types. In other words, AirDAS data must be processed and then concatenated.

AirDAS data is event-based, meaning most events indicate when a state or weather condition changes. For instance, a 'W' event indicates when one or more weather conditions (such as Beaufort sea state) change, and the weather conditions are the same for subsequent events until the next 'W' event. For each state/condition: a new column is created, the state/condition information is extracted from relevant events, and extracted information is propagated to appropriate subsequent rows (events). Thus, each row in the output data frame contains all pertinent state/condition information for that row.

The following assumptions/decisions are made during processing:

- All '#' events (deleted events) are removed
- 'DateTime', 'Lat', and 'Lon' information are added to '1' events where applicable
- Effort is determined as follows: T/R events turns effort on, and O/E events turn effort off. T/R events themselves will be on effort, while O/E events will be off effort. The 'EffortDot' column is ignored
- 'HKR' values are converted to lower case. "Y" values are considered to be "H" values
- Observer ('ObsL', 'ObsB', 'ObsR', 'Rec') values are converted to lower case
- Viewing condition ('VLI', 'VLO', 'VB', 'VRI', 'VRO') values are converted to lower case
- Missing values are NA rather than -1

Normally, a T event (to indicate starting/resuming a transect) is immediately followed by a VPAW event series, creating a TVPAW event series. The reset.transect argument causes the conditions set in the VPAW event series (Beaufort, viewing conditions, altitude, etc.) to be reset to NA at each T event

Value

An airdas_df object, which is also a data frame. It consists of the input data frame, i.e. the output of airdas_read, with the following columns added:

State/condition	*Column name*	*Notes*
On/off effort	OnEffort	
Transect code	Trans	
Beaufort sea state	Bft	
Percent overcast (cloud cover)	CCover	
Jellyfish code	Jelly	not in PHOCOENA data
Horizontal sun (clock system)	HorizSun	
Vertical sun (clock system)	VertSun	only in PHOCOENA data
Haze/Kelp/Red tide code	HKR	
Haze (from HKR code)	Haze	
Kelp (from HKR code)	Kelp	
Red tide (from HKR code)	RedTide	

Altitude (feet)	AltFt
Speed (knots)	SpKnot
Left observer	ObsL
Belly observer	ObsB
Right observer	ObsR
Data recorder	Rec
Viewing condition - left inside	VLI
Viewing condition - left outside	VLO
Viewing condition - belly	VB
Viewing condition - right inside	VRI
Viewing condition - right outside	VRO

See airdas_format_pdf for which data columns the condition information is extracted form for each file type. In addition, warnings are printed with line numbers of unexpected event codes

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
airdas_process(y, trans.upper = FALSE)

y.read <- airdas_read(y)
airdas_process(y.read)
```

`airdas_read` *Read AirDAS file(s)*

Description

Read one or more fixed-width aerial survey DAS text file(s) generated by TURTLEP, or another AirDAS program, into a data frame, where each line is data for a specific event

Usage

```
airdas_read(
  file,
  file.type = c("turtle", "caretta", "survey", "phocoena"),
  skip = 0,
  tz = "UTC",
  ...
)
```

Arguments

file	filename(s) of one or more AirDAS files
file.type	character; indicates the program used to create file. Must be one of: "turtle", "caretta", "survey", or "phocoena" (case sensitive). Default is "turtle"
skip	integer: see read_fwf. Default is 0
tz	character; see strptime. Default is UTC
...	ignored

Details

Reads/parses aerial survey DAS data into columns of a data frame. If file contains multiple filenames, then the individual data frames will be combined using rbind

See airdas_format_pdf for information about AirDAS format requirements for the specific file types (programs)

Value

An airdas_dfr object, which is also a data frame, with AirDAS data read into columns. The data are read into the data frame as characters, with the following exceptions:

Name	*Class*	*Details*
EffortDot	logical	TRUE if "." was present, and FALSE otherwise
DateTime	POSIXct	combination of 'Date' and 'Time' columns, with time zone tz
Lat	numeric	'Latitude' columns converted to decimal degrees in range [-90, 90]
Lon	numeric	'Longitude' columns converted to decimal degrees in range [-180, 180]
Data#	character	leading/trailing whitespace trimmed for non-comment events (i.e. where 'Event' is not "C")
file_das	character	base filename, extracted from the file argument
line_num	integer	line number of each data row
file_type	character	file.type argument

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
airdas_read(y, file.type = "turtle")
```

---

airdas_segdata *Summarize AirDAS data for a continuous effort section*

Description

Summarize AirDAS effort data by effort segment, while averaging conditions

Usage

```
airdas_segdata(x, ...)

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

## S3 method for class 'airdas_df'
airdas_segdata(
  x,
  conditions,
  segdata.method = c("avg", "maxdist"),
  seg.lengths,
  section.id,
  ...
)
```

Arguments

x	airdas_df object, or a data frame that can be coerced to a airdas_df object. Must contain a single continuous effort section of AirDAS data; see the Details section below
...	ignored
conditions	see airdas_effort, or see Details section for more information
segdata.method	character; either "avg" or "maxdist". "avg" means the condition values will be calculated as a weighted average by distance, while "maxdist" means the condition values will be those recorded for the longest distance during that segment
seg.lengths	numeric; length of the modeling segments into which x will be chopped
section.id	numeric; the ID of x (the current continuous effort section)

Details

WARNING - do not call this function directly! It is exported for documentation purposes, but is intended for internal package use only.

This function was designed to be called by one of the airdas_chop_ functions, e.g. airdas_chop_equallength, and thus users should avoid calling it themselves. It loops through the events in x, calculating and storing relevant information for each modeling segment as it goes. Because x is a continuous effort section, it must begin with a "T" or "R" event and end with the corresponding "E" or "O" event.

For each segment, this function reports the segment ID, transect code, the start/end/mid coordinates (lat/lon), start/end/mid date/times (DateTime), segment length, year, month, day, time, observers, and average conditions (which are specified by conditions). The segment ID is designated as section.id _ index of the modeling segment. Thus, if section.id is 1, then the segment ID for the second segment from x is "1_2".

When segdata.method is "avg", the condition values are calculated as a weighted average by distance. The reported value for logical columns (e.g. Haze) is the percentage (in decimals) of the segment in which that condition was TRUE. For character columns, the reported value for each segment

is the unique value(s) present in the segment, with NAs omitted, pasted together via `paste(..., collapse = "; ")`. When `segdata.method` is `"maxdist"`, the reported values are, for each condition, the value recorded for the longest distance during that segment (with NAs omitted).

Transect code, file name, and vent code that started the continuous effort section are also included in the segdata output. These values (excluding NAs) must be consistent across the entire effort section, and thus across all segments in `x`; a warning is printed if there are any inconsistencies.

`bearing` and `destination` are used to calculate the segment start, mid, and end points, with `method = "vincenty"`.

Value

Data frame with the segdata information described above and in `airdas_effort`

`airdas_sight` *Aerial DAS sightings*

Description

Extract sighting information from aerial DAS data

Usage

```
airdas_sight(x, ...)

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

## S3 method for class 'airdas_df'
airdas_sight(x, angle.min = 12, ...)
```

Arguments

`x`	`airdas_df` object; output from `airdas_process`, or a data frame that can be coerced to a `airdas_df` object
`...`	ignored
`angle.min`	numeric; the minimum (absolute value) angle for which to consider a sighting a standard sighting. Default is 12

Details

AirDAS events contain specific information in the 'Data#' columns, with the information depending on the event code and file type for that row. This function extracts relevant data for sighting events, and returns a data frame with dedicated columns for each piece of sighting information. It can handle multiple file types in `x`; for instance, `x` could be processed PHOCOENA and TURTLE data combined using `rbind`. See `airdas_format_pdf` for more information about the expected events and event formats, depending on the file type.

All species codes are converted to lower case using `tolower`.

Abbreviations used in column names include: Gs = group size, Sp = species, Mixed = mixed species (multi-species) sighting. In addition, note that multi-species group sizes are rounded to the nearest whole number using `round(, 0)`

A 'sighting by a standard observer' ('ObsStd') is a sighting made by ObsL, ObsB, or ObsR (not the data recorder or pilot). A 'standard sighting' ('SightStd') is a sighting that was made while on effort, by a standard observer, and with the absolute value of the angle of declination being greater than or equal to `angle.min`. Resights (Events 's') are not considered standard events, and thus both 'ObsStd' and 'SightStd' will be `NA` for 's' events.

Value

Data frame with 1) the columns from `x`, excluding the 'Data#' columns, and 2) columns with sighting information extracted from 'Data#' columns as described below. The data frame has one row for each sighting, or one row for each species of each sighting if it is a multi-species (mixed) sighting.

Added sighting information columns:

Sighting information	*Column name*	*Notes*
Sighting number	SightNo	
Observer that made the sighting	Obs	
Angle of declination	Angle	Left is negative
Sighting by standard observer	ObsStd	Logical; described in Details
Standard sighting	SightStd	Logical; described in Details
Mixed species sighting	Mixed	Logical
Species code	SpCode	All characters converted to lower case
Group size of school	GsTotal	Only different from GsSp for mixed species sightings
Group size of species	GsSp	
Turtle length (feet if numeric)	TurtleSize	`NA` for non-"t" events; may be character or numeric
Turtle travel direction (degrees)	TurtleDirection	`NA` for non-"t" events
Turtle tail visible?	TurtleTail	`NA` for non-"t" events

The TurtleSize will be of class character is there is any CARETTA data in `x`, and of class numeric otherwise.

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.proc <- airdas_process(y)

airdas_sight(y.proc)
```

`as_airdas_df` *Coerce object to a airdas_df object*

Description

Check if an object is of class airdas_df, or coerce it if possible.

Usage

```
as_airdas_df(x)

## S3 method for class 'airdas_df'
as_airdas_df(x)

## S3 method for class 'data.frame'
as_airdas_df(x)
```

Arguments

x	An object to be coerced to class airdas_df

Details

Currently only data frames can be coerced to an object of class airdas_df. If x does not have column names, classes, and contents as specified in airdas_df, then the function returns an error message detailing the first column that does not meet the airdas_df requirements.

Value

An object of class airdas_df

See Also

airdas_df-class

---

as_airdas_dfr *Coerce object to a airdas_dfr object*

Description

Check if an object is of class airdas_dfr, or coerce it if possible.

Usage

```
as_airdas_dfr(x)

## S3 method for class 'airdas_dfr'
as_airdas_dfr(x)

## S3 method for class 'data.frame'
as_airdas_dfr(x)
```

Arguments

x	An object to be coerced to class airdas_dfr

Details

Currently only data frames can be coerced to an object of class airdas_dfr. If x does not have column names and classes as specified in airdas_dfr, then the function returns an error message detailing the first column that does not meet the airdas_dfr requirements.

Value

An object of class 'airdas_dfr'

See Also

airdas_dfr-class

---

subsetting *Subsetting objects created using swfscAirDAS*

Description

Subsetting airdas_dfr or airdas_df objects

Usage

```
## S3 method for class 'airdas_dfr'
x[i, j, ..., drop = TRUE]

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

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

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

## S3 method for class 'airdas_df'
x[i, j, ..., drop = TRUE]

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

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

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

Arguments

x	object of class airdas_dfr or airdas_df
i, j, ...	elements to extract or replace, see [.data.frame
drop	logical, see [.data.frame
name	A literal character string or ..., see [.data.frame
value	A suitable replacement value, see [.data.frame

Details

When subsetting a airdas_dfr or airdas_df object, henceforth a airdas_ object, using any of the functions described in [.data.frame, then then the airdas_ class is simply dropped and the object is of class data.frame. This is because of the strict format requirements of airdas_ objects; it is likely that a subsetted airdas_ object will not have the format required by subsequent swfscAirDAS functions, and thus it is safest to drop the airdas_ class. If a data frame is passed to downstream swfscAirDAS functions that require a airdas_ object, then they will attempt to coerce the object to the necessary airdas_ class See as_airdas_dfr and as_airdas_df for more details.

Examples

```
y <- system.file("airdas_sample.das", package = "swfscAirDAS")
y.read <- airdas_read(y)

# All return a data frame:
class(y.read[1:10, ])
class(y.read[, 1:10])

y.df <- y.read
y.df[, 1] <- "a"
class(y.df)

y.df <- y.read
y.df$Event <- "a"
class(y.df)

y.df <- y.read
y.df[["Event"]] <- "a"
class(y.df)
```

Index