Package 'ARUtools'

April 18, 2024

Type Package

Title Management and Processing of Autonomous Recording Unit (ARU) Data

Version 0.6.2

Description Parse Autonomous Recording Unit (ARU) data and for sub-sampling recordings. Extract Metadata from your recordings, select a subset of recordings for interpretation, and prepare files for processing on the 'WildTrax' <https://wildtrax.ca/> platform. Read and process metadata from recordings collected using the SongMeter and BAR-LT types of ARUs.

License MIT + file LICENSE

URL https://arutools.github.io/ARUtools/, https://github.com/ARUtools/ARUtools

BugReports https://github.com/ARUtools/ARUtools/issues

Depends R (>= 4.0)

Imports dplyr, fs (>= 1.6.1), glue, here, hms (>= 1.1.2), lifecycle, lubridate, lutz, parzer, purrr, readr, rlang (>= 0.4), seewave (>= 2.2.3), sf, spsurvey (>= 5.0.1), stringr, suncalc, tidyr, units, withr

Suggests covr, ggplot2, job, jsonlite, knitr, parallel, patchwork, readxl (>= 1.4.2), rmarkdown, rstudioapi, sessioninfo, soundecology, testthat (>= 3.0.0), tuneR, vdiffr (>= 1.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

NeedsCompilation no

Author David Hope [aut, cre] (<https://orcid.org/0000-0002-2140-4261>), Steffi LaZerte [aut] (<https://orcid.org/0000-0002-7690-8360>), Government of Canada [cph, fnd]

Maintainer David Hope <david.hope@ec.gc.ca>

Repository CRAN

Date/Publication 2024-04-18 19:32:51 UTC

R topics documented:

`acoustic_indices` *Get acoustic complexity values*

Description

Wrapper for 'soundecology' package to calculate acoustic complexity, the bioacoustic index, and acoustic diversity. See Value for details about these indices.

Usage

```
acoustic_indices(
  path,
  min_freq = NA,
  max_freq = NA,
  units = "samples",
  quiet = FALSE
)
```

Arguments

`path`	Character. Path to wave file.
`min_freq`	Numeric. Minimum frequency for acoustic complexity (see `soundecology::acoustic_complexity()`)
`max_freq`	Numeric. Maximum frequency for acoustic complexity (see `soundecology::acoustic_complexity()`)
`units`	Character. Wave file units for reading the file. Defaults to "samples" (see `tuneR::readWave()`).
`quiet`	Logical. Whether to suppress progress messages and other non-essential updates.

Value

Returns a data frame with acoustic indices. Those prefaced with

- `complx_` are from `soundecology::acoustic_complexity()`
- `bio_` are from `soundecology::bioacoustic_index()`
- `div_` are from `soundecology::acoustic_diversity()`

Examples

```
w <- tuneR::sine(440, duration = 300000) # > 5s
tuneR::writeWave(w, "test_wave.wav")
acoustic_indices("test_wave.wav")
acoustic_indices("test_wave.wav", quiet = TRUE)
unlink("test_wave.wav")
```

`add_sites` *Add site-level data to the metadata*

Description

Uses dates to join site-level data (coordinates and site ids) to the meta data. If the site data have only single dates, then a buffer before and after is used to determine which recordings belong to that site observation. Can join by site ids alone if set `by_date = NULL`.

Usage

```
add_sites(
  meta,
  sites,
  buffer_before = 0,
  buffer_after = NULL,
  by = c("site_id", "aru_id"),
  by_date = "date_time",
  quiet = FALSE
)
```

Arguments

`meta`	Data frame. Recording metadata. Output of `clean_metadata()`.
`sites`	Data frame. Site-level data from `clean_site_index()`.
`buffer_before`	Numeric. Number of hours before a deployment in which to include recordings. `NULL` means include the time up to the last deployment. Coupled with `buffer_after`, this creates a window around a date/time in which to join recordings to the site-level data. Ignored if `sites` has both a start and end column for date/times. Default 0.
`buffer_after`	Numeric. Number of hours after the deployment in which to include recordings. `NULL` means include the time up to the next deployment. Coupled with `buffer_before`, creates a window around a date/time in which to join recordings to the site-level data. Ignored if `sites` has both a start and end column for date/times. Default `NULL`.
`by`	Character. Columns which identify a deployment in `sites` as well as `meta`, besides date/time, which are used to join the data. Default is `site_id` and `aru_id`.
`by_date`	Character. Date/time type to join data by. `date` is faster but `date_time` is more precise. Default `date_time`. `NULL` means ignore dates and join only with `by` columns (`dplyr::left_join()`).
`quiet`	Logical. Whether to suppress progress messages and other non-essential updates.

Value

A data frame of metadata with site-level data joined in.

Examples

```
m <- clean_metadata(project_files = example_files)
s <- clean_site_index(example_sites_clean,
  name_date = c("date_time_start", "date_time_end")
)
m <- add_sites(m, s)

# Without dates (by site only)
m <- clean_metadata(project_files = example_files)
eg <- dplyr::select(example_sites_clean, -date_time_start, -date_time_end)
s <- clean_site_index(eg, name_date_time = NULL)
m <- add_sites(m, s, by_date = NULL)
```

`add_wildtrax` *Add file name formated for Wildtrax to metadata*

Description

Create and append file name appropriate for uploading data to the Wildtrax platform https://wildtrax.ca/.

Usage

```
add_wildtrax(meta)
```

Arguments

`meta`	Data frame. Recording metadata. Output of `clean_metadata()`.

Value

Data frame of metadata with appended column of WildTrax appropriate file names.

Examples

```
m <- clean_metadata(project_files = example_files)
m <- add_wildtrax(m)
m
```

ARUtools *ARUtools: Management and Processing of Autonomous Recording Unit (ARU) Data*

Description

Parse Autonomous Recording Unit (ARU) data and for sub-sampling recordings. Extract Metadata from your recordings, select a subset of recordings for interpretation, and prepare files for processing on the WildTrax https://wildtrax.ca/ platform. Read and process metadata from recordings collected using the Song Meter and BAR-LT types of ARUs.

Author(s)

Maintainer: David Hope <david.hope@ec.gc.ca> (ORCID)

Authors:

- Steffi LaZerte <sel@steffilazerte.ca> (ORCID)

Other contributors:

- Government of Canada [copyright holder, funder]

See Also

Useful links:

- https://arutools.github.io/ARUtools/
- https://github.com/ARUtools/ARUtools
- Report bugs at https://github.com/ARUtools/ARUtools/issues

calc_selection_weights *Calculate Selection Weights*

Description

Calculate selection weights for a series of recordings based on the selection parameters defined by `sim_selection_weights()`.

Usage

```
calc_selection_weights(
  meta_sun,
  params,
  col_site_id = site_id,
  col_min = t2sr,
  col_day = date
)
```

Arguments

meta_sun	(Spatial) Data frame. Recording meta data with time to sunrise/sunset. Output of calc_sun(). Must have at least col_min, col_day, and col_site_id.
params	Named list. Parameters created by sim_selection_weights(), containing min_range, min_mean, min_sd, day_range, day_mean, day_sd, offset, return_log, selection_fun.
col_site_id	Column. Unquoted column containing site strata IDs (defaults to site_id).
col_min	Column. Unquoted column containing minutes to sunrise (t2sr) or sunset (t2ss) output from calc_sun() (defaults to t2sr).
col_day	Column. Unquoted column containing dates or day-of-year (doy) to use (defaults to date).

Value

Returns data with appended selection weights columns:

- psel_by - The minutes column used
- psel_min - Probability of selection by time of day (min column)
- psel_doy - Probability of selection by day of year
- psel - Probability of selection overall
- psel_scaled - Probability of selection scaled overall
- psel_std - Probability of selection standardized within a site
- psel_normalized - Probability of selection normalized within a site

Examples

```
s <- clean_site_index(example_sites_clean,
  name_date_time = c("date_time_start", "date_time_end")
)
m <- clean_metadata(project_files = example_files) |>
  add_sites(s) |>
  calc_sun()

params <- sim_selection_weights()
calc_selection_weights(m, params = params)
```

---

calc_sun *Calculate time to sunrise/sunset*

Description

Calculate the sunrise/sunset of each sound file for the day of, the day before and the day after to get the nearest sunrise to the recording. Times are calculated using the 'suncalc' package.

Usage

```
calc_sun(meta_sites, aru_tz = "local")
```

Arguments

meta_sites	(Spatial) Data frame. Recording metadata with added coordinates. Output of clean_metadata() and then add_sites() (with either clean_gps() or clean_site_index()).
aru_tz	Character. Must be either "local" or a timezone listed in OlsonNames(). See Details.

Details

Timezones. To ensure that the sunrise/sunset times are calculated correctly relative to the time of the recording, we need to know the timezone of the date/time of the recording. If ARUs were calibrated with a specific timezone before going into the field, that can be specified by using, for example, aru_tz = "America/Toronto". If on the other hand each ARU was calibrated to whichever timezone was local when it was deployed use aru_tz = "local". The specific timezone will be calculated individually based on the longitude and latitude of each recording.

Value

Data frame with metadata and added timezone of recording time (tz), and time to sunrise/sunset (t2sr, t2ss).

Examples

```
s <- clean_site_index(example_sites_clean,
  name_date = c("date_time_start", "date_time_end")
)
m <- clean_metadata(project_files = example_files) |>
  add_sites(s)
calc_sun(m)
```

---

check_file *Explore a file*

Description

Shows the first few lines in a text file. Useful for trying to understand problems in GPS files.

Usage

```
check_file(file_name, n_max = 10, ...)
```

Arguments

file_name	Character. File path to check.
n_max	Numeric. Number of lines in the file to show. Default 10.
...	Arguments passed on to readr::read_lines()

Details

Wrapper around `readr::read_lines(n_max)`.

Value

A character vector with one element for each line

Examples

```
f <- system.file("extdata", "logfile_00015141_SD1.txt", package = "ARUtools")
check_file(f)
```

---

check_meta *Check output of* `clean_metadata()`

Description

Cleaning metadata can take a series of tries. This function helps summarize and explore the metadata for possible patterns which may help find problems.

Usage

```
check_meta(meta, date = FALSE)
```

Arguments

`meta`	Data frame. Recording metadata. Output of `clean_metadata()`.
`date`	Logical. Whether to summarize output by date (as well as `site_id` and `aru_id`. Default `FALSE`.

Value

A data frame summarizing the metadata by site_id, aru_type, aru_id, and (optionally) by date. Presents the number of files, directories, and days worth of recordings, as well as the minimum and maximum recording times.

Examples

```
m <- clean_metadata(project_files = example_files)

check_meta(m)
check_meta(m, date = TRUE)
```

check_problems *Check problems in output of* clean_metadata()

Description

Cleaning metadata can take a series of tries. This function helps summarize and explore missing metadata (problems).

Usage

```
check_problems(
  df,
  check = c("site_id", "aru_id", "date", "date_time", "longitude", "latitude"),
  path = FALSE,
  date = FALSE
)
```

Arguments

df	Data frame. Either meta data (clean_metadata()) or GPS coordinates (clean_gps())
check	Character. Character vector of columns to check for missing values. Default is site_id, aru_id, date, date_time, longitude and latitude.
path	Logical. Whether to return just the file paths which have missing attributes. Default FALSE
date	Logical. Whether to summarize output by date (as well as site_id and aru_id. Default FALSE.

Value

A data frame summarizing the metadata by site_id, aru_type, aru_id, and (optionally) by date. Presents the number of files, directories, and days worth of recordings, as well as the minimum and maximum recording times.

Examples

```
m <- clean_metadata(project_files = example_files, pattern_aru_id = "test")

check_problems(m)
check_problems(m, date = TRUE)
check_problems(m, path = TRUE)
```

clean_gps *Check and clean GPS data*

Description

Check and clean GPS data from ARU logs. GPS points are checked for obvious problems (expected range, distance cutoffs and timing) then attached to the meta data frame. Note that it is often safer and more reliable to create your own Site Index file including site ids, and GPS coordinates. This file can be cleaned and prepared with `clean_site_index()` instead.

Usage

```
clean_gps(
  meta = NULL,
  dist_cutoff = 100,
  dist_crs = 3161,
  dist_by = c("site_id", "aru_id"),
  quiet = FALSE,
  verbose = FALSE
)
```

Arguments

`meta`	Data frame. Output of `clean_metadata()`.
`dist_cutoff`	Numeric. Maximum distance (m) between GPS points within a site. Default is 100m but can be set to `Inf` to skip.
`dist_crs`	Numeric. Coordinate Reference System to use when calculating distance (should be one with m).
`dist_by`	Character. Column which identifies sites within which to compare distance among GPS points. Only valid if `dist_cutoff` is not `Inf`.
`quiet`	Logical. Whether to suppress progress messages and other non-essential updates.
`verbose`	Logical. Show extra loading information. Default `FALSE`.

Details

If checking for a maximum distance (`dist_cutoff`) among GPS points within a group (`dist_by`), the returned data frame will include a column `max_dist`, which represents the largest distance among points within that group.

Value

Data frame of site-level metadata.

Examples

```
m <- clean_metadata(project_dir = "my_project")
g <- clean_gps(meta = m)
```

`clean_logs` *Extract log data from BAR-LT log files*

Description

Process BAR-LT log files into a data frame reflecting metadata, schedule information, and events. Events are time-stamped logs of either GPS fixes (`lat` and `lon`) or recordings (`rec_file`, `rec_size`, `rec_end`).

Usage

```
clean_logs(
  log_files,
  return = "all",
  pattern_sr = "(SR)",
  pattern_ss = "(SS)",
  progress = TRUE
)
```

Arguments

`log_files`	Character vector of log files to process.
`return`	Character. What kind of data to return, GPS fixes (`"gps"`), recording events (`"recordings"`) or `"all"` (default).
`pattern_sr`	Character. Pattern to match the sunrise schedule in the log files.
`pattern_ss`	Character. Pattern to match the sunset schedule in the log files.
`progress`	Logical. Whether to use `purrr::map()` progress bars (default `TRUE`).

Details

Note that log files can have glitches. If there is no start time for a recording (generally when there is a problem and no recording is made), the `date_time` value for that recording will be the same as the `rec_end` time.

Because the BAR-LT units adjust their time according to the GPS locations, all times are in "local" to that area.

Value

Data frame containing

- `file_names` and `paths` of the log files
- `events` and their `date_times`
- `lat` and `lon` for "gps" events
- `rec_file`, `rec_size` and `rec_end` for "recording" events (recording start is the `date_time` of the event)
- `schedule` information such as `schedule_date`, `schedule_name`, `schedule_lat`, `schedule_lon`, `schedule_sr` (sunrise), and `schedule_ss` (sunset)
- `metadata` information such as `meta_serial` and `meta_firmware`

Examples

```
# Replace "my_project_folder" with your directory containing your recordings and logfiles
log_files <- fs::dir_ls("my_project_folder", recurse = TRUE, glob = "*logfile*")
log_files
logs <- clean_logs(log_files)

log_files <- "../ARUtools - Extra/aru_log_files/P028/1A_BARLT10962/logfile_00010962_SD1.txt"

clean_logs(log_files)
clean_logs(log_files, return = "gps")
clean_logs(log_files, return = "recordings")

log_files <- fs::dir_ls("../ARUtools - Extra/aru_log_files/", recurse = TRUE, glob = "*logfile*")

l <- clean_logs(log_files)
```

---

clean_metadata *Extract and clean ARU metadata from file names*

Description

Using regular expressions, metadata is extracted from file names and directory structure, checked and cleaned.

Usage

```
clean_metadata(
  project_dir = NULL,
  project_files = NULL,
  file_type = "wav",
  subset = NULL,
  subset_type = "keep",
```

```
  pattern_site_id = create_pattern_site_id(),
  pattern_aru_id = create_pattern_aru_id(),
  pattern_date = create_pattern_date(),
  pattern_time = create_pattern_time(),
  pattern_dt_sep = create_pattern_dt_sep(),
  order_date = "ymd",
  quiet = FALSE
)
```

Arguments

project_dir	Character. Directory where project files are stored. File paths will be used to extract information and must actually exist.
project_files	Character. Vector of project file paths. These paths can be absolute or relative to the working directory, and don't actually need to point to existing files unless you plan to use clean_gps() or other sampling steps down the line. Must be provided if project_dir is NULL.
file_type	Character. Type of file (extension) to summarize. Default wav.
subset	Character. Text pattern to mark a subset of files/directories to either "keep" or "omit" (see subset_type)
subset_type	Character. Either keep (default) or omit files/directories which match the pattern in subset.
pattern_site_id	Character. Regular expression to extract site ids. See create_pattern_site_id(). Can be a vector of multiple patterns to match.
pattern_aru_id	Character. Regular expression to extract ARU ids. See create_pattern_aru_id(). Can be a vector of multiple patterns to match.
pattern_date	Character. Regular expression to extract dates. See create_pattern_date(). Can be a vector of multiple patterns to match.
pattern_time	Character. Regular expression to extract times. See create_pattern_time(). Can be a vector of multiple patterns to match.
pattern_dt_sep	Character. Regular expression to mark separators between dates and times. See create_pattern_dt_sep().
order_date	Character. Order that the date appears in. "ymd" (default), "mdy", or "dmy". Can be a vector of multiple patterns to match.
quiet	Logical. Whether to suppress progress messages and other non-essential updates.

Details

Note that times are extracted by first combining the date, date/time separator and the time patterns. This means that if there is a problem with this combination, dates might be extracted but date/times will not. This mismatch can be used to determine which part of a pattern needs to be tweaked.

See vignette("customizing", package = "ARUtools") for details on customizing clean_metadata() for your project.

Value

Data frame with extracted metadata

Examples

```
clean_metadata(project_files = example_files)
clean_metadata(project_files = example_files, subset = "P02")
```

`clean_site_index` *Prepare and clean site index file*

Description

A site index file contains information on when specific ARUs were deployed where. This function cleans a file (csv, xlsx) or data frame in preparation for adding these details to the output of `clean_metadata()`. It can be used to specify missing information according to date, such as GPS lon/lats and site ids.

Usage

```
clean_site_index(
  site_index,
  name_aru_id = "aru_id",
  name_site_id = "site_id",
  name_date_time = "date",
  name_coords = c("longitude", "latitude"),
  name_extra = NULL,
  resolve_overlaps = TRUE,
  quiet = FALSE
)
```

Arguments

`site_index`	(Spatial) Data frame or file path. Site index data to clean. If file path, must be to a local csv or xlsx file.
`name_aru_id`	Character. Name of the column that contains ARU ids. Default `"aru_id"`.
`name_site_id`	Character. Name of the column that contains site ids. Default `"site_id"`.
`name_date_time`	Character. Column name that contains dates or date/times. Can be vector of two names if there are both 'start' and 'end' columns. Can be `NULL` to ignore dates. Default `"date"`.
`name_coords`	Character. Column names that contain longitude and latitude (in that order). Ignored if `site_index` is spatial. Default `c("longitude", "latitude")`
`name_extra`	Character. Column names for extra data to include. If a named vector, will rename the columns (see examples). Default `NULL`.

`resolve_overlaps`
: Logical. Whether or not to resolve date overlaps by shifting the start/end dates to noon (default `TRUE`). This assumes that ARUs are generally *not* deployed/removed at midnight (the official start/end of a day) and so noon is used as an approximation for when an ARU was deployed or removed. If possible, use specific deployment times to avoid this issue.

`quiet`
: Logical. Whether to suppress progress messages and other non-essential updates.

Details

Note that times are assumed to be in 'local' time and a timezone isn't used (and is removed if present, replaced with UTC). This allows sites from different timezones to be processed at the same time.

Value

Standardized site index data frame

Examples

```
s <- clean_site_index(example_sites,
  name_aru_id = "ARU",
  name_site_id = "Sites",
  name_date_time = c("Date_set_out", "Date_removed"),
  name_coords = c("lon", "lat")
)

s <- clean_site_index(example_sites,
  name_aru_id = "ARU",
  name_site_id = "Sites",
  name_date_time = c("Date_set_out", "Date_removed"),
  name_coords = c("lon", "lat"),
  name_extra = c("plot" = "Plots")
)

# Without dates
eg <- dplyr::select(example_sites, -Date_set_out, -Date_removed)
s <- clean_site_index(eg,
  name_aru_id = "ARU",
  name_site_id = "Sites",
  name_date_time = NULL,
  name_coords = c("lon", "lat"),
  name_extra = c("plot" = "Plots")
)
```

clip_wave *Clip multiple wave files and format names*

Description

Process multiple wave files by copying them with a new filename and clipping to a given length.

Usage

```
clip_wave(
  waves,
  dir_out,
  dir_in = NULL,
  col_path_in = path,
  col_subdir_out = subdir_out,
  col_filename_out = filename_out,
  col_clip_length = clip_length,
  col_start_time = start_time,
  overwrite = FALSE,
  create_dir = TRUE,
  diff_limit = 30,
  use_job = FALSE
)
```

Arguments

waves	Data frame. Details of file locations.
dir_out	Character. Output directory.
dir_in	Character. Directory wave files are read from. Default is NULL meaning the current working directory.
col_path_in	Column. Unquoted column containing the current file paths. Default path. **Note: file paths must be either relative to** dir_in **or absolute**.
col_subdir_out	Column. Unquoted column containing the subdirectories in which to put output files. Default subdir_out.
col_filename_out	Column. Unquoted column containing the output filenames. Default filename_out.
col_clip_length	Column. Unquoted column containing the length of the new clip. Default length.
col_start_time	Column. Unquoted column containing the start time of the new clip. Default start_time.
overwrite	Logical. Overwrite pre-existing files when clipping and moving. Default FALSE.
create_dir	Logical. Whether to create directory structure for newly formatted and clipped wave files.

diff_limit	Numeric. How much longer in seconds clip lengths can be compared to file lengths before triggering an error. Default 30.
use_job	Logical. Use the 'job' package to copy files Default FALSE.

Value

TRUE if successful and clipped wave files created

Examples

```
w <- data.frame(
  path = temp_wavs(n = 4),
  subdir_out = c("test1/a", "test2/a", "test3/c", "test4/d"),
  subsub_dir_out = rep("zz", 4),
 filename_out = c("wave1_clean.wav", "wave2_clean.wav", "wave3_clean.wav", "wave4_clean.wav"),
  clip_length = c(1, 1, 1, 2),
  start_time = c(1.2, 0.5, 1, 0)
)

clip_wave(w, dir_out = "clean", col_subdir_out = c(subdir_out, subsub_dir_out))

unlink("clean", recursive = TRUE) # Remove this new 'clean' directory
```

---

clip_wave_single *Clip single wave file*

Description

Clip and copy a single wave files to a given length. See clip_wave() for processing multiple files.

Usage

```
clip_wave_single(
  path_in,
  path_out,
  clip_length,
  start_time = 0,
  wave_length = NULL,
  overwrite = FALSE
)
```

Arguments

path_in	Character. Path to the wave file to clip.
path_out	Character. Path to copy the new clipped wave file to.
clip_length	Numeric. Length of new clip in seconds.

start_time	Numeric. Time in seconds where new clip should start. Default 0.
wave_length	Numeric. Length of the clipped wave file in seconds (if NULL, default, will be the length of time from start_time to the end of the file).
overwrite	Logical. Whether to overwrite existing files when creating new clipped wave files. Default (FALSE) will error if the file already exists.

Value

TRUE if successful

Examples

```
# Create test wave file
f <- temp_wavs(1)

# Clip file and check it out
clip_wave_single(f, "new_file.wav", clip_length = 1)
tuneR::readWave("new_file.wav")
unlink("new_file.wav")
```

---

count_files *Count files in a project directory*

Description

Helper function to explore the number of files in a directory, recursively.

Usage

```
count_files(project_dir, subset = NULL, subset_type = "keep")
```

Arguments

project_dir	Character. Directory where project files are stored. File paths will be used to extract information and must actually exist.
subset	Character. Text pattern to mark a subset of files/directories to either "keep" or "omit" (see subset_type)
subset_type	Character. Either keep (default) or omit files/directories which match the pattern in subset.

Value

A data frame with number of files in a directory

Examples

```
count_files("PROJECT_DIR")
```

`create_dirs` *Create directory structure for recording folders*

Description

Create a set of nested folders for storing ARU recordings by plots and sites.

Usage

```
create_dirs(
  plots,
  site_ids,
  base_dir = NULL,
  dir_list = FALSE,
  dry_run = TRUE,
  expect_dirs = FALSE
)
```

Arguments

`plots`	Character vector. Hexagon or cluster names for folder names.
`site_ids`	Character vector. Site IDs. Should include the plot/cluster id in the name.
`base_dir`	Character. Base directory to build directory structure in.
`dir_list`	Logical. Whether to return a vector of directories (to be) created (defaults to `FALSE`).
`dry_run`	Logical. Whether to do a dry-run of the process (i.e. do not actually create directories; defaults to `TRUE`)
`expect_dirs`	Logical. Expect that directories may already exist? Default (`FALSE`) is to stop if directories to be created already exist.

Value

If `dir_list = TRUE`, returns a list of directories (to be) created. If not a dry run, also creates the folder structure.

Examples

```
# Default is to do a dry-run (don't actually create the directories)
create_dirs(
  plots = c("river1", "river2", "river3"),
  site_ids = c(
    "river1_sm01", "river1_sm02", "river2_sm03", "river2_sm04",
    "river3_sm05", "river3_sm06"
  ),
  base_dir = "Recordings"
)
```

```
# Get a list of directories which would be created
create_dirs(
  plots = c("river1", "river2", "river3"),
  site_ids = c(
    "river1_sm01", "river1_sm02", "river2_sm03", "river2_sm04",
    "river3_sm05", "river3_sm06"
  ),
  base_dir = "Recordings", dir_list = TRUE
)


# Create directories AND return a list of those created
d <- create_dirs(
  plots = c("river1", "river2", "river3"),
  site_ids = c(
    "river1_sm01", "river1_sm02", "river2_sm03", "river2_sm04",
    "river3_sm05", "river3_sm06"
  ),
  base_dir = "Recordings", dir_list = TRUE, expect_dirs =TRUE,
  dry_run = FALSE
)
d
```

`create_pattern` *Create a pattern to match date*

Description

Helper functions to create regular expression patterns to match different metadata in file paths.

Usage

```
create_pattern_date(order = "ymd", sep = c("_", "-", ""), yr_digits = 4)

create_pattern_time(sep = c("_", "-", ":", ""), seconds = "yes")

create_pattern_dt_sep(sep = "T", optional = FALSE)

create_pattern_aru_id(
  arus = c("BARLT", "S\\d(A|U)", "SM\\d", "SMM", "SMA"),
  n_digits = c(4, 8),
  sep = c("_", "-", ""),
  prefix = "",
  suffix = ""
)

create_pattern_site_id(
  prefix = c("P", "Q"),
```

```
  p_digits = 2,
  sep = c("_", "-"),
  suffix = "",
  s_digits = 1
)

test_pattern(test, pattern)
```

Arguments

order	Character vector. Expected orders of (y)ear, (m)onth and (d)ate. Default is "ymd" for Year-Month-Date order. Can have more than one possible order.
sep	Character vector. Expected separator(s) between the pattern parts. Can be "" for no separator.
yr_digits	Numeric vector. Number of digits in Year, either 2 or 4.
seconds	Character. Whether seconds are included. Options are "yes", "no", "maybe".
optional	Logical. Whether the separator should be optional or not. Allows matching on different date/time patterns.
arus	Character vector. Pattern(s) identifying the ARU prefix (usually model specific).
n_digits	Numeric vector. Number of digits expected to follow the `arus` pattern. Can be one or two (a range).
prefix	Character vector. Prefix(es) for site ids.
suffix	Character vector. Suffix(es) for site ids.
p_digits	Numeric vector. Number(s) of digits following the `prefix`.
s_digits	Numeric vector. Number(s) of digits following the `suffix`.
test	Character vector. Examples of text to test.
pattern	Character. Regular expression pattern to test.

Details

By default `create_pattern_aru_id()` matches many common ARU patterns like `BARLT0000`, `S4A0000`, `SM40000`, `SMM0000`, `SMA0000`.

`test_pattern()` is a helper function to see what a regular expression pattern will pick out of some example text. Can be used to see if your pattern grabs what you want. This is just a simple wrapper around `stringr::str_extract()`.

Value

Either a pattern (`create_pattern_xxx()`) or the text extracted by a pattern (`test_pattern()`)

Functions

- `create_pattern_date()`: Create a pattern to match a date
- `create_pattern_time()`: Create a pattern to match a time
- `create_pattern_dt_sep()`: Create a pattern to match a date/time separator

- `create_pattern_aru_id()`: Create a pattern to match an ARU id
- `create_pattern_site_id()`: Create a pattern to match a site id
- `test_pattern()`: Test patterns

Examples

```
create_pattern_date() # Default matches 2020-01-01 or 2020_01_01 or 20200101
# ("-", "_" or "" as separators)
create_pattern_date(sep = "") # Matches only 20200101 (no separator allowed)

create_pattern_time() # Default matches 23_59_59 (_, -, :, as optional separators)
create_pattern_time(sep = "", seconds = "no") # Matches 2359 (no seconds no separators)

create_pattern_dt_sep() # Default matches 'T' as a required separator
create_pattern_dt_sep(optional = TRUE) # 'T' as an optional separator
create_pattern_dt_sep(c("T", "_", "-")) # 'T', '_', or '-' as separators

create_pattern_aru_id()
create_pattern_aru_id(prefix = "CWS")
create_pattern_aru_id(n_digits = 12)


create_pattern_site_id() # Default matches P00-0
create_pattern_site_id(
  prefix = "site", p_digits = 3, sep = "",
  suffix = c("a", "b", "c"), s_digits = 0
) # Matches site000a

pat <- create_pattern_aru_id(prefix = "CWS")
test_pattern("CWS_BARLT1012", pat) # No luck
pat <- create_pattern_aru_id(prefix = "CWS_")
test_pattern("CWS_BARLT1012", pat) # Ah ha!
pat <- create_pattern_site_id()

pat <- create_pattern_site_id()
test_pattern("P03", pat) # Nope
test_pattern("P03-1", pat) # Success!

pat <- create_pattern_site_id(prefix = "site", p_digits = 3, sep = "", s_digits = 0)
test_pattern("site111", pat)
pat <- create_pattern_site_id(
  prefix = "site", p_digits = 3, sep = "",
  suffix = c("a", "b", "c"), s_digits = 0
)
test_pattern(c("site9", "site100a"), pat)
```

`example_clean` *Example cleaned recording meta data*

Description

A data frame with examples of correctly formatted metadata with added site-level information

Usage

```
example_clean
```

Format

`example_clean`:
A data frame with 42 rows and 10 columns:

file_name Name of the file
type File type
path Relative file path including file name
aru_type ARU model
aru_id ARU ids
site_id Site ids
date_time Recording date/time
date Recording date
longitude Latitude in decimal degrees
latitude Longitude in decimal degrees

Source

data-raw/data_test.R

---

example_files *Example recording files*

Description

A vector of examples ARU recording files.

Usage

```
example_files
```

Format

`example_files`:
A vector with 42 file paths

Source

data-raw/data_test.R

example_files_long *Example long-term deployment recording files*

Description

A vector of examples ARU recording files. Uses the `example_sites` data, but deploys them for a longer deployment

Usage

```
example_files_long
```

Format

`example_files_long`**:**
A vector with 614 file paths

Source

data-raw/data_long_deployment.R

example_sites *Example site-level meta data*

Description

A data frame with examples of incorrectly formatted site-level data.

Usage

```
example_sites
```

Format

`example_sites`**:**
A data frame with 10 rows and 8 columns:

Sites Site ids
Date_set_out Deployment start date
Date_removed Deployment end date
ARU ARU ids
lon Longitude in decimal degrees
lat Latitude in decimal degrees
Plots Hypothetical extra plot column
Subplot Hypothetical extra subplot column

Source

data-raw/data_test.R

`example_sites_clean` *Example cleaned site-level meta data*

Description

A data frame with examples of correctly formatted site-level data.

Usage

```
example_sites_clean
```

Format

`example_sites_clean`**:**
A data frame with 10 rows and 8 columns:

site_id Site ids
aru_id ARU ids
date_time_start Deployment start date/time
date_time_end Deployment end date/time
date_start Deployment start date
date_end Deployment end date
longitude Latitude in decimal degrees
latitude Longitude in decimal degrees

Source

data-raw/data_test.R

`get_wav_length` *Get the length of a recording in seconds*

Description

Get the length of a recording in seconds

Usage

```
get_wav_length(path, return_numeric = FALSE)
```

Arguments

path	Character. Path to wave file.
return_numeric	Logical. Return numeric or character?

Value

Length of recording in seconds

Examples

```
f <- tempfile()
w <- tuneR::sine(440, duration = 100000)
tuneR::writeWave(w, f)
get_wav_length(f)
```

---

sample_recordings *Sample recordings*

Description

Sample recordings based on selection weights from `calc_selection_weights()` using `spsurvey::grts()`.

Usage

```
sample_recordings(
  meta_weights,
  n,
  os = NULL,
  col_site_id = site_id,
  col_sel_weights = psel_std,
  seed = NULL,
  ...
)
```

Arguments

meta_weights	(Spatial) Data frame. Recording meta data selection weights. Output of `calc_selection_weights()`. Must have at least the columns identified by `col_site_id` and `col_sel_weights`, as well as the probability of selection columns (those starting with `psel`) and `doy`.
n	Numeric, Data frame, Vector, or List. Number of base samples to choose. For stratification by site, a named vector/list of samples per site, or a data frame with columns `n` for samples, `n_os` for oversamples and the column matching that identified by `col_site_id`.
os	Numeric, Vector, or List. Over sample size (proportional) or named vector/list of number of samples per site Ignored if `n` is a data frame.

col_site_id	Column. Unquoted column containing site strata IDs (defaults to site_id).
col_sel_weights	Column. Unquoted name of column identifying selection weights (defaults to psel_std)
seed	Numeric. Random seed to use for random sampling. Seed only applies to specific sampling events (does not change seed in the environment). NULL does not set a seed.
...	Extra named arguments passed on to spsurvey::grts().

Value

A sampling run from grts. Note that the included dataset is spatial, but is a dummy spatial dataset created by using dates and times to create the spatial landscape.

Examples

```
s <- clean_site_index(example_sites_clean,
  name_date_time = c("date_time_start", "date_time_end")
)
m <- clean_metadata(project_files = example_files) |>
  add_sites(s) |>
  calc_sun()

params <- sim_selection_weights()
w <- calc_selection_weights(m, params = params)

# No stratification by site
samples <- sample_recordings(w, n = 10, os = 0.1, col_site_id = NULL)

# Stratification by site defined by...

# lists
samples <- sample_recordings(w, n = list(P01_1 = 2, P02_1 = 5, P03_1 = 2), os = 0.2)

# vectors
samples <- sample_recordings(w, n = c(P01_1 = 2, P02_1 = 5, P03_1 = 2), os = 0.2)

# data frame
samples <- sample_recordings(
  w,
  n = data.frame(
    site_id = c("P01_1", "P02_1", "P03_1"),
    n = c(2, 5, 2),
    n_os = c(0, 0, 1)
  )
)
```

`sim_selection_weights` *Create parameters and simulate selection weights*

Description

This function creates and explores parameters for generating selections. These parameters define the selection distribution of minutes (`min`) around the sun event (sunrise/sunset), as well as of days (`day`).

Usage

```
sim_selection_weights(
  min_range = c(-70, 240),
  min_mean = 30,
  min_sd = 60,
  day_range = c(120, 201),
  day_mean = 161,
  day_sd = 20,
  offset = 0,
  return_log = TRUE,
  selection_fun = "norm",
  selection_var = "psel_normalized",
  return_params = TRUE,
  plot = TRUE
)
```

Arguments

`min_range`	Numeric vector. Range of the sampling distribution of minutes around the sun event.
`min_mean`	Numeric. Mean of the sampling distribution of minutes to the sun event.
`min_sd`	Numeric. SD in minutes of the sampling distribution of minutes around the sun event.
`day_range`	Date/Datetime/Numeric vector. Range of sampling distribution of days. Can be Dates, Date-times, or DOY (day-of-year, 1-366).
`day_mean`	Date/Datetime/Numeric. Mean date of the sampling distribution of days. Can be Date, Date-time, or DOY (day-of-year, 1-366).
`day_sd`	Numeric. SD in days of the sampling distribution of days.
`offset`	Numeric. Offset to shift for time of day in minutes.
`return_log`	Logical. Log the density in the selection function?
`selection_fun`	Character. Selection function to use. Options are `lognorm`, `norm` (default), or `cauchy`.
`selection_var`	Character. Selection variable to plot (if `plot = TRUE`). Options are are `psel`, `psel_doy`, `psel_min`, `psel_std`, `psel_scaled`, or `psel_normalized` (default).

return_params	Logical. Return parameter list for use in calc_selection_weights()?
plot	Logical. Create plot of simulated selection weights? If return_param = TRUE and plot = TRUE plot is created as a side effect. Other wise, plot is returned directly.

Value

Returns either a list of selection parameters or a plot of simulated selection weights

Examples

```
params <- sim_selection_weights()
```

---

sox_spectro *Create spectrogram image from wave file*

Description

Using the external program SoX (the Swiss Army knife of sound processing programs), create a spectrogram image file. Note that you must have SoX installed to use this function. Spectrograms will be silently overwritten.

Usage

```
sox_spectro(
  path,
  dir_out = "Spectrograms",
  prepend = "spectro_",
  width = NULL,
  height = NULL,
  start = NULL,
  end = NULL,
  rate = "20k",
  dry_run = FALSE,
  quiet = FALSE,
  sox_file_path = NULL,
  skip_check = FALSE
)
```

Arguments

path	Character. Path to wave file.
dir_out	Character. Output directory.
prepend	Character. Text to add to the start of the output file. Defaults to "spectro_".
width	Numeric. Width of the spectrogram image in pixels.
height	Numeric. Height of the spectrogram image in pixels.

start	Numeric/Character. Start the spectrogram at this time (seconds or HH:MM:SS format).
end	Numeric/Character. End time the spectrogram at this time (seconds or HH:MM:SS format).
rate	Numeric. Audio sampling rate to display (used by the rate effect in sox). This effectively limits the upper frequency of the spectrogram to rate/2. The default ("20k"), limits the spectrogram to 10kHz. Use rate = NULL for no limiting.
dry_run	Logical. If TRUE show the sox command, but do not run (for debugging and understanding precise details).
quiet	Logical. Whether to suppress progress messages and other non-essential updates.
sox_file_path	Path to sox file if not installed at the system level, otherwise NULL.
skip_check	Logical. Should the function skip check to ensure SoX is installed. This may allow speed ups if running across large numbers of files.

Details

Most arguments are passed through to the seewave::sox() command.

- width and height correspond to the -x and -y options for the spectrogram effect.
- start and end are used by the trim effect
- rate is passed on to the rate effect

Based on code from Sam Hache.

Value

Does not return anything, but creates a spectrogram image in dir_out.

Examples

```
# Prep sample file
w <- tuneR::sine(440, duration = 300000)
td <- tempdir()
temp_wave <- glue::glue("{td}/test_wave.wav")
tuneR::writeWave(w, temp_wave)

# Create spectrograms

try({sox_spectro(temp_wave)
sox_spectro(temp_wave, rate = NULL)
sox_spectro(temp_wave, start = 2, end = 3)
sox_spectro(temp_wave, start = "0:01", end = "0:04")
sox_spectro(temp_wave, prepend = "")
})

# Clean up
unlink(temp_wave)
unlink("Spectrograms", recursive = TRUE)
```

task_template *Example template of tasks for WildTrax*

Description

A data frame with tasks generated from `example_clean` using the wildRtrax::wt_make_aru_tasks() function. Allows updating of tasks on WildTrax https://wildtrax.ca/.

Usage

```
task_template
```

Format

`task_template`**:**

A data frame with 14 rows and 13 columns:

location Site location name

recording_date_time Date time of the recording

method Method of interpretation (generally '1SPT')

taskLength Length of recording in seconds

transcriber Transcriber ID, to be filled in with function

rain Empty character for filling in WildTrax

wind Empty character for filling in WildTrax

industryNoise Empty character for filling in WildTrax

audioQuality Empty character for filling in WildTrax

taskComments Empty character for filling in WildTrax

internal_task_id Empty character for filling in WildTrax

Source

data-raw/data_wt_assign_tasks.R

template_observers *Example template of tasks for WildTrax*

Description

A data frame showing example observers and their effort

Usage

```
template_observers
```

Format

`template_observers`**:**

A data frame with 4 rows and 2 columns:

transcriber Interpreter name in Wildtrax system

hrs Number of hours to assign to interpreter

Source

data-raw/data_wt_assign_tasks.R

`temp_wavs` *Helper function to create test wave files*

Description

Creates a directory structure and example wave files in temp folders.

Usage

```
temp_wavs(n = 6)
```

Arguments

`n`	Numeric. How many test files to create (up to six). D

Value

vector of paths to temporary wave files

Examples

```
temp_wavs(n=3)
```

`wind_detection_pre_processing` *Pre-processing of files for Wind Detection program*

Description

[Experimental]

This function takes a vector of wave file names and returns a list of three vectors that can be provided to the wind detection software or written to files that the software can read. Details of the usable fork of the wind detection software can be found at https://github.com/dhope/WindNoiseDetection

Usage

```
wind_detection_pre_processing(
  wav_files,
  site_pattern,
  output_directory,
  write_to_file = FALSE,
  chunk_size = NULL
)
```

Arguments

`wav_files`	Vector of path to wav files
`site_pattern`	Pattern to extract sites from file names
`output_directory`	Directory path to export files to
`write_to_file`	Logical Should the function write files to output_directory
`chunk_size`	Numeric If not NULL, sets number of files to include in each chunk

Value

List including filePath, filenames, and sites suitable for wind software.

Examples

```
wind_files <-
wind_detection_pre_processing(
wav_files = example_clean$path,
  output_directory = td,
    site_pattern = create_pattern_site_id(
        p_digits = c(2, 3), sep = "_",
            s_digits = c(1, 2)
              ),
                write_to_file = FALSE, chunk_size = NULL
                )
```

`wind_detection_summarize_json` *Summarize wind detection results*

Description

[Experimental]

This function takes output from the command line program and summarizes it. Details of the wind detection software can be found at https://github.com/dhope/WindNoiseDetection.

Usage

```
wind_detection_summarize_json(f)
```

Arguments

f	filepath for json
	#'

Value

tibble of summarized data from json file

Examples

```
# example code

 example_json <- system.file("extdata",
 "P71-1__20210606T232500-0400_SS.json",
 package = "ARUtools"
 )

 wind_summary <- wind_detection_summarize_json(example_json)
```

`wt_assign_tasks` *Assign tasks for interpretation on Wildtrax*

Description

Assign tasks for interpretation on Wildtrax

Usage

```
wt_assign_tasks(
  wt_task_template_in,
  interp_hours,
  wt_task_output_file,
  interp_hours_column,
  random_seed = NULL
)
```

Arguments

`wt_task_template_in`	Path to csv template downloaded from Wildtrax platform https://wildtrax.ca listing all tasks. Alternatively, can be a data.frame that is correctly formatted using `wildRtrax::wt_make_aru_tasks()`. See `vignette("Misc")` for details.
`interp_hours`	Path to number of hours for each interpreter or a data.table. If a file, must be csv and must include the columns "transcriber" and whatever the variable `interp_hours_column` is.
`wt_task_output_file`	Path to csv of output file for uploading to Wildtrax. If left as NULL will not write file
`interp_hours_column`	LazyEval column name with hours for interpreters
`random_seed`	Integer. Random seed to select with. If left NULL will use timestamp

Value

Returns a list with a tibble of assigned tasks and a summary tibble.

Examples

```
task_output <- wt_assign_tasks(
wt_task_template_in = task_template,
wt_task_output_file = NULL,
interp_hours = template_observers,
interp_hours_column = hrs,
random_seed = 65122
)
```

Index