

Package: camtraptor (via r-universe)

July 1, 2026

Title Read, Explore and Visualize Camera Trap Data Packages

Version 0.28.0

Description Read, explore and visualize Camera Trap Data Packages (Camtrap DP). 'Camtrap DP' (<<https://camtrap-dp.tdwg.org>>) is a community developed data exchange format for this type of data. With camtraptor you can read and filter data, create overviews of observed species, relative abundance or effort, and plot these data on a map.

License MIT + file LICENSE

BugReports <https://github.com/inbo/camtraptor/issues>

Depends R (>= 3.5.0)

Imports activity, assertthat, dplyr (>= 1.1.0), EML, frictionless, glue, htmltools, leaflet, lifecycle, lubridate, overlap, purrr, RColorBrewer, readr, rlang, reshape2, stringr, tidyr, tools, utils, uuid

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), xml2

VignetteBuilder knitr

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/pak/sysreqs

libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libjq-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libnode-dev libx11-dev

Repository <https://camera-traps.r-universe.dev>

Date/Publication 2026-04-28 07:05:55 UTC

RemoteUrl <https://github.com/inbo/camtraptor>

RemoteRef HEAD

RemoteSha 0d86eacec14e9c59f5e734f382e385a722acd003

Contents

animal_positions	2
apply_filter_predicate	3
calc_animal_pos	4
check_species	5
dep_calib_models	6
get_cam_op	7
get_custom_effort	8
get_detection_history	11
get_effort	15
get_n_individuals	16
get_n_obs	18
get_n_species	19
get_rai	20
get_rai_individuals	22
get_record_table	23
get_scientific_name	26
get_species	27
map_dep	28
mica	34
pred	35
read_camtrap_dp	39
read_wi	40
round_coordinates	41
write_dwc	42
write_eml	43
Index	46

animal_positions *Sample of animal position digitization data*

Description

A tibble data frame with the following columns:

- deploymentID
- sequenceID
- x and y: The coordinates.
- imageWidth and imageHeight: The image dimensions.

Usage

animal_positions

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 42 rows and 6 columns.

See Also

Other sample data: [dep_calib_models](#), [mica](#)

`apply_filter_predicate`

Intermediate function to apply filter predicates on a data frame

Description

This function is used internally by all the `get_*()` functions to filter on deployments.

Usage

```
apply_filter_predicate(df, verbose, ...)
```

Arguments

<code>df</code>	Data frame we want to apply filter(s) expression(s)
<code>verbose</code>	Show (TRUE) or not (FALSE) the filter predicate expression.
<code>...</code>	filter predicates to apply to <code>df</code>

Value

A data frame.

See Also

Other filter functions: [pred\(\)](#)

Examples

```
# and
apply_filter_predicate(
  mica$data$deployments,
  verbose = TRUE,
  pred_gte("latitude", 51.28),
  pred_lt("longitude", 3.56)
)
# Equivalent of
apply_filter_predicate(
  mica$data$deployments,
  verbose = TRUE,
  pred_and(
    pred_gte("latitude", 51.28),
```

```

    pred_lt("longitude", 3.56)
  )
)

# or
apply_filter_predicate(
  mica$data$deployments,
  verbose = TRUE,
  pred_or(
    pred_gte("latitude", 51.28),
    pred_lt("longitude", 3.56)
  )
)

```

calc_animal_pos	<i>Calculate animal position</i>
-----------------	----------------------------------

Description

Calculates the position of animal relative to a camera based on image pixel positions and site calibration models.

Usage

```

calc_animal_pos(
  animal_pos,
  calib_models,
  dep_tag = "deploymentID",
  sequence_id = "sequenceID",
  x = "x",
  y = "y",
  image_width = "imageWidth",
  image_height = "imageHeight"
)

```

Arguments

animal_pos	Data frame (tibble) of animal position digitization data. It must contain (at least) the columns defined in args dep_tag, sequence_id, x, y, image_width and image_height.
calib_models	Named list of deployment calibration models or site calibration models (calibs objects), produced using cal.site() (not yet included in this package). The deployment names are used as names.
dep_tag	Column in animal_pos against which names of the elements can be matched to apply the right deployment calibration models. Default: "deploymentID".
sequence_id	Column in animal_pos containing the sequence ID the images belong to. Default: "sequenceID".

x	Column in animal_pos containing x pixel positions for each digitised point. Default: "x".
y	Column in animal_pos containing y pixel positions for each digitised point. Default: "y".
image_width	Column in animal_pos containing the pixel x dimension of each image. Default: "imageWidth". Notice that the pixel x dimension must be consistent for each deployment.
image_height	Column in animal_pos containing the pixel y dimension of each image. Default: "imageHeight". Notice that the pixel y dimension must be consistent for each deployment.

Value

Original tibble data frame as passed via animal_pos with additional columns:

- radius: Radial distance from camera.
- angle: Angular distance from camera.
- frame_count: Indicator of the frame order within each sequence.

Examples

```
# Use default values
calc_animal_pos(animal_positions, dep_calib_models)
```

check_species	<i>Check scientific or vernacular name(s)</i>
---------------	---

Description

Checks if a given scientific or vernacular name(s) can be found in the metadata (package\$taxonomic) and returns error if not.

Usage

```
check_species(
  package = NULL,
  species,
  arg_name = "species",
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
species	Character vector with scientific or vernacular names.
arg_name	Character with argument name to return in error message Default: "species".
datapkg	Deprecated. Use package instead.

Value

A character vector with the correspondent scientific names.

Examples

```
# Species is a scientific name
check_species(mica, "Martes foina")

# Species is a vector of vernacular names
check_species(mica, c("beech marten", "european polecat"))

# Vernacular names can be specified in any language available
check_species(mica, c("vos", "blauwe reiger"))

# Vernacular names and scientific names can be mixed up
check_species(mica, c("beech marten", "blauwe reiger", "Anas strepera"))

# Case insensitive
check_species(mica, "AnaS StrePeRa")
check_species(mica, "bEEch mARteN")

## Not run:
check_species(mica, "bad name")

## End(Not run)
```

dep_calib_models

Sample of deployment calibration models

Description

A list containing a number of calibration models (calibs) or site calibration models (depcal). The deployment names are used as names.

Usage

```
dep_calib_models
```

Format

An object of class calibs of length 4.

See Also

Other sample data: [animal_positions](#), [mica](#)

get_cam_op	<i>Get camera operation matrix</i>
------------	------------------------------------

Description

Returns the **camera operation matrix** as returned by `camtrapR::cameraOperation()`.

Usage

```
get_cam_op(
  package = NULL,
  ...,
  station_col = "locationName",
  camera_col = NULL,
  session_col = NULL,
  use_prefix = FALSE,
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by <code>read_camtrap_dp()</code> .
...	filter predicates for filtering on deployments.
station_col	Column name to use for identifying the stations. Default: "locationName".
camera_col	Column name of the column specifying Camera ID. Default: NULL.
session_col	Column name to use for identifying the session. Default: NULL. Use it for creating multi-session / multi-season detection histories.
use_prefix	Logical (TRUE or FALSE). If TRUE the returned row names will start with prefix "Station" as returned by <code>camtrapR::cameraOperation()</code> . Default: FALSE.
datapkg	Deprecated. Use package instead.

Details

The deployment data are by default grouped by `locationName` (station ID in `camtrapR` jargon) or another column specified by the user via the `station_col` argument. If multiple deployments are linked to same location, daily efforts higher than 1 occur.

Partially active days, e.g. the first or the last day of a deployment, result in decimal effort values as in `camtrapR::cameraOperation()`.

Value

A matrix. Row names always indicate the station ID. Column names are dates.

See Also

Other `camtrapR`-derived functions: [get_detection_history\(\)](#), [get_record_table\(\)](#)

Examples

```

library(dplyr)
get_cam_op(mica)

# Applying filter(s) on deployments, e.g. deployments with latitude >= 51.18
get_cam_op(mica, pred_gte("latitude", 51.18))

# Specify column with station names
get_cam_op(mica, station_col = "locationID")

# Specify column with session IDs
mica_sessions <- mica
mica_sessions$data$deployments <- mica_sessions$data$deployments %>%
  dplyr::mutate(session = ifelse(
    stringr::str_starts(.data$locationName, "B_DL_"),
    "after2020",
    "before2020"
  )
)
get_cam_op(mica_sessions, session_col = "session")

# Specify column with camera IDs
mica_cameras <- mica_sessions
mica_cameras$data$deployments$cameraID <- c(1, 2, 3, 4)
get_cam_op(mica_cameras, camera_col = "cameraID")

# Specify both session and camera IDs
get_cam_op(mica_cameras, camera_col = "cameraID", session_col = "session")

# Use prefix Station as in camtrapR's camera operation matrix
get_cam_op(mica, use_prefix = TRUE)

```

```

get_custom_effort      Get custom effort

```

Description

Gets the effort for each deployment and a specific time interval such as day, week, month or year. A custom time window can also be set up. This function calls `get_cam_op()` internally.

Usage

```

get_custom_effort(
  package = NULL,
  ...,
  start = NULL,
  end = NULL,
  group_by = NULL,
  unit = "hour",

```

```

  datapkg = lifecycle::deprecated()
)

```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
...	filter predicates
start	Start date. Default: NULL. If NULL the earliest start date among all deployments is used. If group_by unit is not NULL, the lowest start value allowed is one group by unit before the start date of the earliest deployment. If this condition doesn't hold true, a warning is returned and the earliest start date among all deployments is used. If group_by unit is NULL the start must be later than or equal to the start date among all deployments.
end	End date. Default: NULL. If NULL the latest end date among all deployments is used. If group_by unit is not NULL, the latest end value allowed is one group by unit after the end date of the latest deployment. If this condition doesn't hold true, a warning is returned and the latest end date among all deployments is used. If group_by unit is NULL the end must be earlier than or equal to the end date among all deployments.
group_by	Character, one of "day", "week", "month", "year". The effort is calculated at the interval rate defined in group_by. Default: NULL: no grouping, i.e. the entire interval from start to end is taken into account as a whole. Calendar values are used, i.e. grouping by year will calculate the effort from Jan 1st up to Dec 31st for each year.
unit	Character, the time unit to use while returning custom effort. One of: hour (default), day.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with following columns:

- deploymentID: Deployment unique identifier.
- locationName: Location name of the deployments.
- begin: Begin date of the interval the effort is calculated over.
- effort: The effort as number.
- unit: Character specifying the effort unit.

See Also

Other exploration functions: [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```

# Effort for each deployment over the entire duration of the project
# (datapackage) measured in hours (default)
get_custom_effort(mica)

# Effort for each deployment expressed in days
get_custom_effort(mica, unit = "day")

# Effort for each deployment from a specific start to a specific end
get_custom_effort(
  mica,
  start = as.Date("2019-12-15"), # or lubridate::as_date("2019-12-15")
  end = as.Date("2021-01-10")
)

# Effort for each deployment at daily interval
get_custom_effort(
  mica,
  group_by = "day"
)

# Effort for each deployment at weekly interval
get_custom_effort(
  mica,
  group_by = "week"
)

# Effort for each deployment at monthly interval
get_custom_effort(
  mica,
  group_by = "month"
)

# Effort for each deployment at yearly interval
get_custom_effort(
  mica,
  group_by = "year"
)

# Applying filter(s), e.g. deployments with latitude >= 51.18, can be
# combined with other arguments
get_custom_effort(mica, pred_gte("latitude", 51.18), group_by = "month")

# You can afterwards calculate the total effort over all deployments
library(dplyr)
get_custom_effort(mica, group_by = "year", unit = "day") %>%
  dplyr::filter(effort > 0) %>%
  dplyr::group_by(begin) %>%
  dplyr::summarise(
    deploymentIDs = list(deploymentID),
    locationNames = list(locationName),
    ndep = length(unique(deploymentID)),

```

```

    nloc = length(unique(locationName)),
    effort = sum(effort),
    unit = unique(unit)
  )

```

get_detection_history *Get the detection history of a species*

Description

This function creates a detection history matrix for a species based on the record table and the camera operation matrix. The detection history matrix is a binary matrix where rows represent camera stations and columns represent occasions. The matrix is filled with 1s and 0s, where 1 indicates that the species was detected at a station on a given occasion and 0 indicates that the species was not detected. The function also returns the effort matrix, which contains the number of days that each station was active on each occasion, and the dates matrix, which contains the dates of the occasions.

Usage

```

get_detection_history(
  recordTable,
  camOp,
  species,
  output,
  occasionLength = 1,
  minActiveDaysPerOccasion = NULL,
  maxNumberDays = NULL,
  day1 = "station",
  buffer = NULL,
  unmarkedMultFrameInput = FALSE
)

```

Arguments

recordTable	A data frame with the camera trap records. The data frame should contain the columns 'Station', 'Date', 'Species' and 'n'. 'Station' is the camera station ID, 'Date' is the date of the record, 'Species' is the species name, 'n' is the number of observations, and 'n_ind' is the number of individuals detected.
camOp	A matrix with camera operation data. Rows represent camera stations and columns represent occasions. The matrix should contain the number of days that each station was active on each occasion.
species	Character. The species name.
output	Character. The type of output. Choose one of: "binary", "n_observations", "n_individuals".
occasionLength	Integer. The length of the occasions in days. No decimals allowed. Default: 1.

<code>minActiveDaysPerOccasion</code>	Integer. Minimum number of active trap days for occasions to be included. Default: NULL. If used, it must be smaller than or equal to <code>occasionLength</code> .
<code>maxNumberDays</code>	Integer. Maximum number of trap days per station. Default: NULL. If used, it must be greater than or equal to <code>occasionLength</code> .
<code>day1</code>	Character. Day occasions should begin: station setup date ("station") or a specific date (e.g. "2015-12-31"). For multi-season detection history (<code>unmarkedMultFrameInput = TRUE</code>), only <code>day1 = "station"</code> is allowed. Default: "station".
<code>buffer</code>	Integer. It makes the first occasion begin a number of days after station setup. <code>buffer</code> can be used only in combination with <code>day1 = "station"</code> . Default: NULL. A warning is returned if some records are removed because taken during the buffer period.
<code>unmarkedMultFrameInput</code>	Logical. If TRUE, the function will return the input for multi-season occupancy models in unmarked R package (argument <code>y</code> in <code>unmarked::unmarkedMultFrame()</code>). Default: FALSE.

Details

This function doesn't take as input a camera trap data package object, but a camera operation matrix and a record table, which are both calculated based on a camera trap data package object. For more information, see the `get_cam_op()` and `get_record_table()` functions.

If the camera operation matrix (`camOp`) was created for a multi-season study (via argument `session_col` in `get_cam_op()`), the session will be detected automatically. You can then set `unmarkedMultFrameInput = TRUE` to generate a multi-season detection history. Each row corresponds to a site, and the columns are in season-major, occasion-minor order, e.g. `o1_SESS_A`, `o2_SESS_A`, `o1_SESS_B`, `o2_SESS_B`, etc.

Value

A list with three elements:

- `detection_history`: the detection history matrix
- `effort`: the effort matrix
- `dates`: the dates matrix

See Also

Other `camtrapR`-derived functions: `get_cam_op()`, `get_record_table()`

Examples

```
library(dplyr)
camOp <- get_cam_op(mica)
recordTable <- get_record_table(mica)

# Binary output
get_detection_history(
```

```
    recordTable,
    camOp,
    species = "Anas platyrhynchos",
    output = "binary"
  )

# Number of observations output
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_observations"
)

# Number of individuals output
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_individuals"
)

# Occasion length of 7 days
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_individuals",
  occasionLength = 7
)

# Use a `minActiveDaysPerOccasion` of 5 days
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_individuals",
  occasionLength = 7,
  minActiveDaysPerOccasion = 5
)

# Use a `maxNumberDays` of 5 days
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_individuals",
  maxNumberDays = 5
)

# Specify start date via `day1`
get_detection_history(
  recordTable,
```

```

    camOp,
    species = "Anas platyrhynchos",
    output = "binary",
    day1 = "2020-06-22"
  )

# Use a `buffer` of 2 days
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_individuals",
  buffer = 2
)

# Multi-season detection history

# Create a multi-season camera operation matrix / record table
mica_sessions <- mica
mica_sessions$data$deployments$session <- c("2020", "2020", "2021", "2021")
mica_sessions$data$deployments$locationID <- c(
  mica_sessions$data$deployments$locationID[1:2],
  mica_sessions$data$deployments$locationID[1:2]
)
mica_sessions$data$deployments$locationName <- c(
  mica_sessions$data$deployments$locationName[1:2],
  mica_sessions$data$deployments$locationName[1:2]
)
delta <- lubridate::duration(2, units = "years")
mica_sessions$data$deployments$start[4] <- mica_sessions$data$deployments$start[4] + delta
mica_sessions$data$deployments$end[4] <- mica_sessions$data$deployments$end[4] + delta
mica_sessions$data$observations <- mica_sessions$data$observations %>%
dplyr::mutate(timestamp = dplyr::if_else(
  deploymentID %in% mica_sessions$data$deployments$deploymentID[4],
  timestamp + delta,
  timestamp
)
)
mica_sessions$data$media <- mica_sessions$data$media %>%
dplyr::mutate(
  timestamp = dplyr::if_else(
    deploymentID %in% mica_sessions$data$deployments$deploymentID[4],
    timestamp + delta,
    timestamp
  )
)
camOp_sessions <- get_cam_op(mica_sessions, session_col = "session")
recordTable_sessions <- get_record_table(mica_sessions)

# Create a multi-season detection history
get_detection_history(
  recordTable_sessions,
  camOp_sessions,

```

```

species = "Anas platyrhynchos",
output = "n_individuals",
unmarkedMultFrameInput = TRUE
)

```

get_effort

Get effort

Description

Gets the effort (deployment duration) per deployment.

Usage

```

get_effort(
  package = NULL,
  ...,
  unit = "hour",
  datapkg = lifecycle::deprecated()
)

```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
...	filter predicates
unit	Time unit to use while returning deployment effort (duration). One of: <ul style="list-style-type: none"> • second • minute • hour • day • month • year
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with following columns:

- deploymentID: Deployment unique identifier.
- effort: Effort expressed in the unit passed by parameter unit.
- unit: The unit used to express the effort. One of the values available for parameter unit.
- effort_duration: A duration object (duration is a class from lubridate package).

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```
# Efforts expressed in hours
get_effort(mica)

# Effort expressed as days
get_effort(mica, unit = "day")
```

get_n_individuals	<i>Get number of individuals for each deployment</i>
-------------------	--

Description

Gets the number of individuals (of a subset of species) per deployment. The number of observed individuals is stored in field count of observations.

Usage

```
get_n_individuals(
  package = NULL,
  ...,
  species = "all",
  sex = NULL,
  life_stage = NULL,
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by <code>read_camtrap_dp()</code> .
...	filter predicates for filtering on deployments
species	Character with scientific names or common names (case insensitive). If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
sex	Character defining the sex class to filter on, e.g. "female" or <code>c("male", "unknown")</code> . If NULL (default) all observations of all sex classes are taken into account.
life_stage	Character vector defining the life stage class to filter on, e.g. "adult" or <code>c("subadult", "adult")</code> . If NULL (default) all observations of all life stage classes are taken into account.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with the following columns:

- deploymentID: Deployment unique identifier.
- scientificName: Scientific name of the species. This column is omitted if parameter species = NULL.
- n: Number of individuals.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```
# Get number of observations for each species
get_n_individuals(mica)

# Get number of obs of all species, not identified individuals as well
get_n_individuals(mica, species = NULL)

# Get number of observations of Anas platyrhynchos
get_n_individuals(mica, species = "Anas platyrhynchos")

# Get number of observations of eurasian beaver (vernacular name)
get_n_individuals(mica, species = "eurasian beaver")

# Mix scientific and vernacular names
get_n_individuals(mica, species = c("Anas platyrhynchos", "eurasian beaver"))

# Case insensitive
get_n_individuals(mica, species = "AnAS PLatyrhynchOS")
get_n_individuals(mica, species = "eurasian BEAVER")

# Specify life stage
get_n_individuals(mica, life_stage = "adult")

# Specify sex
get_n_individuals(mica, sex = "female")

# Specify both sex and life stage
get_n_individuals(mica, sex = "unknown", life_stage = "adult")

# Apply filter(s), e.g. deployments with latitude >= 51.18
get_n_individuals(mica, pred_gte("latitude", 51.18))
```

get_n_obs	<i>Get number of observations for each deployment</i>
-----------	---

Description

Gets the number of observations (of a subset of species) per deployment. The number of observations is defined as the number of distinct sequences (sequenceID).

Usage

```
get_n_obs(
  package = NULL,
  ...,
  species = "all",
  sex = NULL,
  life_stage = NULL,
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by <code>read_camtrap_dp()</code> .
...	Filter predicates for filtering on deployments
species	Character with scientific names or common names (case insensitive). If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
sex	Character defining the sex class to filter on, e.g. "female" or <code>c("male", "unknown")</code> . If NULL (default) all observations of all sex classes are taken into account.
life_stage	Character vector defining the life stage class to filter on, e.g. "adult" or <code>c("subadult", "adult")</code> . If NULL (default) all observations of all life stage classes are taken into account.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with the following columns:

- deploymentID: Deployment unique identifier.
- scientificName: Scientific name of the species. This column is omitted if parameter species = NULL.
- n: Number of observations.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```

# Get number of observations for each species
get_n_obs(mica)

# Get number of obs of all species, not identified individuals as well
get_n_obs(mica, species = NULL)

# Get number of observations of Anas platyrhynchos (scientific name)
get_n_obs(mica, species = "Anas platyrhynchos")

# Get number of observations of eurasian beaver (vernacular names)
get_n_obs(mica, species = "eurasian beaver")

# Case insensitive
get_n_obs(mica, species = "Anas plaTYrhYnchoS")
get_n_obs(mica, species = "EUrasian beavER")

# Specify life stage
get_n_obs(mica, life_stage = "subadult")

# Specify sex
get_n_obs(mica, sex = "female")

# Specify both sex and life stage
get_n_obs(mica, sex = "unknown", life_stage = "adult")

# Applying filter(s), e.g. deployments with latitude >= 51.18
get_n_obs(mica, pred_gte("latitude", 51.18))

```

get_n_species	<i>Get number of identified species for each deployment</i>
---------------	---

Description

Gets the number of identified species per deployment.

Usage

```
get_n_species(package = NULL, ..., datapkg = lifecycle::deprecated())
```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
...	Filter predicates for filtering on deployments.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with the following columns:

- deploymentID: Deployment unique identifier.
- n: Number of observed and identified species.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```
# Get number of species
get_n_species(mica)

# Get number of species for deployments with latitude >= 51.18
get_n_species(mica, pred_gte("latitude", 51.18))
```

get_rai

Get Relative Abundance Index (RAI)

Description

Gets the RAI (Relative Abundance Index) per deployment. The RAI is normalized using 100 days deployment activity. In other words: $RAI = 100 * (n/effort)$ where n is the number of observations as calculated via [get_n_obs\(\)](#) and effort is the effort in days as calculated via [get_effort\(\)](#).

Usage

```
get_rai(
  package = NULL,
  ...,
  species = "all",
  sex = NULL,
  life_stage = NULL,
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp() .
...	Filter predicates for filtering on deployments.
species	Character with scientific names or common names (case insensitive). If "all" (default) all scientific names are automatically selected.

sex	Character defining the sex class to filter on, e.g. "female" or c("male", "unknown"). If NULL (default) all observations of all sex classes are taken into account.
life_stage	Character vector defining the life stage class to filter on, e.g. "adult" or c("subadult", "adult"). If NULL (default) all observations of all life stage classes are taken into account.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with the following columns:

- deploymentID: Deployment unique identifier.
- scientificName: Scientific name.
- rai: Relative abundance index.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```
# Calculate RAI for all species
get_rai(mica) # species = "all" by default, so equivalent of
get_rai(mica, species = "all")

# Selected species
get_rai(mica, species = c("Anas platyrhynchos", "Martes foina"))

# With vernacular names, even mixing languages
get_rai(mica, species = c("mallard", "steenmarter"))

# Mixed scientific and vernacular names
get_rai(mica, species = c("Anas platyrhynchos", "steenmarter"))

# Species parameter is case insensitive
get_rai(mica, species = c("ANAS pLAtyRhynChOS"))

# Specify sex
get_rai(mica, sex = "female")
get_rai(mica, sex = c("female", "unknown"))

# Specify life stage
get_rai(mica, life_stage = "adult")
get_rai(mica, life_stage = c("adult", "subadult"))

# Apply filter(s): deployments with latitude >= 51.18
get_rai(mica, pred_gte("latitude", 51.18))
```

get_rai_individuals *Get Relative Abundance Index (RAI) based on number of individuals*

Description

Function to get the RAI (Relative Abundance Index) per deployment based on number of detected individuals instead of the number of observations. The RAI is normalized using 100 days deployment activity. In other words: $RAI = 100 * (n/effort)$ where n is the number of individuals as calculated via `get_n_individuals()` and `effort` is the effort in days as calculated via `get_effort()`.

Usage

```
get_rai_individuals(
  package = NULL,
  ...,
  species = "all",
  sex = NULL,
  life_stage = NULL,
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by <code>read_camtrap_dp()</code> .
...	Filter predicates for filtering on deployments.
species	Character with scientific names or common names (case insensitive). If "all" (default) all scientific names are automatically selected.
sex	Character defining the sex class to filter on, e.g. "female" or <code>c("male", "unknown")</code> . If NULL (default) all observations of all sex classes are taken into account.
life_stage	Character vector defining the life stage class to filter on, e.g. "adult" or <code>c("subadult", "adult")</code> . If NULL (default) all observations of all life stage classes are taken into account.
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with the following columns:

- deploymentID: Deployment unique identifier.
- scientificName: Scientific name.
- rai: Relative abundance index.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_scientific_name\(\)](#), [get_species\(\)](#)

Examples

```

# Calculate RAI based on number of individuals
get_rai_individuals(mica) # species = "all" by default, so equivalent of
get_rai_individuals(mica, species = "all")

# Selected species
get_rai_individuals(mica,
  species = c("Anas platyrhynchos", "Martes foina")
)

# With common names, also mixing up languages
get_rai_individuals(mica, species = c("mallard", "steenmarter"))

# Mixed scientific and vernacular names
get_rai_individuals(mica, species = c("Anas platyrhynchos", "beech marten"))

# Species parameter is case insensitive
get_rai_individuals(mica, species = c("ANAS pLAtyRhynChOS"))

# Specify sex
get_rai_individuals(mica, sex = "female")
get_rai_individuals(mica, sex = c("female", "unknown"))

# Specify life stage
get_rai_individuals(mica, life_stage = "adult")
get_rai_individuals(mica, life_stage = c("adult", "subadult"))

# Apply filter(s): deployments with latitude >= 51.18
get_rai_individuals(mica, pred_gte("latitude", 51.18))

```

<code>get_record_table</code>	<i>Get record table</i>
-------------------------------	-------------------------

Description

Calculates the record table from a camera trap data package and so tabulating species records. The record table is a concept developed within the `camtrapR` package, see [this article](#). See also the function documentation for `camtrapR::recordTable()`. **Note:** All dates and times are expressed in UTC format.

Usage

```

get_record_table(
  package = NULL,
  ...,
  stationCol = "locationName",
  exclude = NULL,
  minDeltaTime = 0,
  deltaTimeComparedTo = NULL,

```

```

removeDuplicateRecords = TRUE,
datapkg = lifecycle::deprecated()
)

```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
...	Filter predicates for filtering on deployments
stationCol	Character name of the column containing stations. Default: "locationName".
exclude	Character vector of species names (scientific names or vernacular names) to be excluded from the record table. Default: NULL.
minDeltaTime	Time difference between records of the same species at the same station to be considered independent (in minutes). Default: 0.
deltaTimeComparedTo	One of "lastIndependentRecord" or "lastRecord". For two records to be considered independent, the second one must be at least minDeltaTime minutes after the last independent record of the same species (deltaTimeComparedTo = "lastIndependentRecord"), or minDeltaTime minutes after the last record (deltaTimeComparedTo = "lastRecord"). If minDeltaTime is 0, deltaTimeComparedTo must be NULL (default).
removeDuplicateRecords	Logical. If there are several records of the same species at the same station at exactly the same time, show only one?
datapkg	Deprecated. Use package instead.

Value

A tibble data frame containing species records and additional information about stations, date, time and further metadata, such as filenames and directories of the images (media) linked to the species records. Some more details about the columns returned:

- **Station:** Character, station names, as found in the deployment column defined in parameter stationCol.
- **Species:** Character, the scientific name of the observed species.
- **n:** Numeric, the number of individuals of the observed species.
- **DateTimeOriginal:** Datetime object, as found in column timestamp of observations, in UTC format.
- **Date:** Date object, the date part of DateTimeOriginal, in UTC format.
- **Time:** Character, the time part of DateTimeOriginal in UTC format.
- **delta.time.secs:** Numeric, the duration in seconds from the previous independent record of a given species at a certain location.
- **delta.time.mins:** Numeric, the duration in minutes from the previous independent record of a given species at a certain location.
- **delta.time.hours:** Numeric, the duration in hours from the previous independent record of a given species at a certain location.

- `delta.time.days`: Numeric, the duration in days from the previous independent record of a given species at a certain location.
- `Directory`: List, file paths of the images linked to the given record, as defined in column `filePath` of `media`.
- `Filename`: List, file names of the images linked to the given record, as defined in column `fileName` of `media`.
- `Latitude`: Numeric, latitude of the station, based on `deploymentID` of the observations.
- `Longitude`: Numeric, longitude of the station, based on `deploymentID` of the observations.
- `clock`: Numeric, clock time in radians.
- `solar`: Numeric, solar time in radians. Calculated using `overlap::sunTime`, which essentially uses the approach described in [Nouvellet et al. \(2012\)](#).

See Also

Other camtrapR-derived functions: [get_cam_op\(\)](#), [get_detection_history\(\)](#)

Examples

```
get_record_table(mica)

# Set a minDeltaTime of 20 minutes from last independent record for filtering
# out not independent observations
mica_dependent <- mica
mica_dependent$data$observations[4,"timestamp"] <- lubridate::as_datetime("2020-07-29 05:55:00")
get_record_table(
  mica_dependent,
  minDeltaTime = 20,
  deltaTimeComparedTo = "lastIndependentRecord"
)

# Set a minDeltaTime of 20 minutes from last record for filtering out not
# independent observations
get_record_table(
  mica_dependent,
  minDeltaTime = 20,
  deltaTimeComparedTo = "lastRecord"
)

# Exclude observations of mallard
# Exclude is case insensitive and vernacular names are allowed
get_record_table(mica, exclude = "wilde eend")

# Specify column to pass station names
get_record_table(
  mica,
  stationCol = "locationID",
  minDeltaTime = 20,
  deltaTimeComparedTo = "lastRecord"
)
```

```

# How to deal with duplicates
mica_dup <- mica
# create a duplicate at 2020-07-29 05:46:48, location: B_DL_val 5_beek kleine vijver
mica_dup$data$observations[4, "sequenceID"] <- mica_dup$data$observations$sequenceID[3]
mica_dup$data$observations[4, "deploymentID"] <- mica_dup$data$observations$deploymentID[3]
mica_dup$data$observations[4, "timestamp"] <- mica_dup$data$observations$timestamp[3]

# duplicates are removed by default by get_record_table()
get_record_table(mica_dup)

# duplicate not removed
get_record_table(mica_dup, removeDuplicateRecords = FALSE)

# Applying filter(s) on deployments, e.g. deployments with latitude >= 51.18
get_record_table(mica, pred_gte("latitude", 51.18))

```

get_scientific_name *Get scientific name for vernacular name*

Description

Gets the scientific name for one or more vernacular names based on the taxonomic information found in the metadata (package\$taxonomic). The match is performed case insensitively. If a vernacular name is not valid, an error is returned

Usage

```

get_scientific_name(
  package = NULL,
  vernacular_name,
  datapkg = lifecycle::deprecated()
)

```

Arguments

package Camera trap data package object, as returned by read_camtrap_dp().

vernacular_name Character vector with input vernacular name(s).

datapkg Deprecated. Use package instead.

Value

Character vector of scientific name(s).

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_species\(\)](#)

Examples

```
# One or more vernacular names
get_scientific_name(mica, "beech marten")
get_scientific_name(mica, c("beech marten", "mallard"))

# Vernacular names can be passed in different languages
get_scientific_name(mica, c("beech marten", "wilde eend"))

# Search is performed case insensitively
get_scientific_name(mica, c("MaLLaRd"))

## Not run:
# An error is returned if at least one invalid vernacular name is passed
get_scientific_name(mica, "this is a bad vernacular name")

# A scientific name is an invalid vernacular name of course
get_scientific_name(mica, c("Castor fiber", "wilde eend"))

## End(Not run)
```

get_species

Get species

Description

Gets all identified species.

Usage

```
get_species(package = NULL, datapkg = lifecycle::deprecated())
```

Arguments

package	Camera trap data package object, as returned by read_camtrap_dp().
datapkg	Deprecated. Use package instead.

Value

A tibble data frame with all scientific names and vernacular names of the identified species.

See Also

Other exploration functions: [get_custom_effort\(\)](#), [get_effort\(\)](#), [get_n_individuals\(\)](#), [get_n_obs\(\)](#), [get_n_species\(\)](#), [get_rai\(\)](#), [get_rai_individuals\(\)](#), [get_scientific_name\(\)](#)

Examples

```
get_species(mica)
```

map_dep	<i>Visualize deployments features</i>
---------	---------------------------------------

Description

This function visualizes deployments features such as number of detected species, number of observations and RAI on a dynamic map. The circle size and colour are proportional to the mapped feature. Deployments without observations are shown as gray circles and a message is returned.

Usage

```
map_dep(
  package = NULL,
  feature,
  ...,
  species = NULL,
  sex = NULL,
  life_stage = NULL,
  effort_unit = NULL,
  cluster = TRUE,
  hover_columns = c("n", "species", "deploymentID", "locationID", "locationName",
    "latitude", "longitude", "start", "end"),
  palette = "inferno",
  zero_values_show = TRUE,
  zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/000000/multiply.png",
  zero_values_icon_size = 10,
  na_values_show = TRUE,
  na_values_icon_url = "https://img.icons8.com/ios-glyphs/30/FA5252/multiply.png",
  na_values_icon_size = 10,
  relative_scale = TRUE,
  max_scale = NULL,
  radius_range = c(10, 50),
  datapkg = lifecycle::deprecated()
)
```

Arguments

package	Camera trap data package object, as returned by <code>read_camtrap_dp()</code> .
feature	Deployment feature to visualize. One of: <ul style="list-style-type: none"> • <code>n_species</code>: Number of identified species. • <code>n_obs</code>: Number of observations. • <code>n_individuals</code>: Number of individuals. • <code>rai</code>: Relative Abundance Index. • <code>rai_individuals</code>: Relative Abundance Index based on number of individuals. • <code>effort</code>: Effort (duration) of the deployment.

...	Filter predicates for subsetting deployments.
species	Character with a scientific name. Required for <code>rai</code> , optional for <code>n_obs</code> . Default: <code>NULL</code> .
sex	Character defining the sex class to filter on, e.g. "female". If <code>NULL</code> (default) all observations of all sex classes are taken into account. Optional parameter for <code>n_obs</code> and <code>n_individuals</code> .
life_stage	Character vector defining the life stage class to filter on, e.g. "adult" or <code>c("subadult", "adult")</code> . If <code>NULL</code> (default) all observations of all life stage classes are taken into account. Optional parameter for <code>n_obs</code> and <code>n_individuals</code> .
effort_unit	Time unit to use while visualizing deployment effort (duration). One of: <ul style="list-style-type: none"> • second • minute • hour • day • month • year If <code>NULL</code> (default), the effort is returned in hours.
cluster	Logical value indicating whether using the cluster option while visualizing maps. Default: <code>TRUE</code> .
hover_columns	Character vector with the name of the columns to use for showing location deployment information on mouse hover. One or more from deployment columns. Use <code>NULL</code> to disable hovering. Default information: <ul style="list-style-type: none"> • <code>n</code>: Number of species, number of observations, RAI or effort (column created internally by a <code>get_*</code>(<code>)</code> function). • <code>species</code>: Species name(s). • <code>start</code>: Start deployment. • <code>end</code>: End deployment. • <code>deploymentID</code>: Deployment unique identifier. • <code>locationID</code>: Location unique identifier. • <code>locationName</code>: Location name. • <code>latitude</code> • <code>longitude</code> See the Deployment section of Camtrap DP for the full list of columns you can use.
palette	The palette name or the colour function that values will be mapped to. Typically one of the following: <ul style="list-style-type: none"> • A character vector of RGB or named colours. Examples: <code>c("#000000", "#0000FF", "#FFFFFF")</code> • The full name of a <code>RColorBrewer</code> palette, e.g. "BuPu" or "Greens", or <code>viridis</code> palette: "viridis", "magma", "inferno" or "plasma". For more options, see parameter <code>palette</code> of <code>leaflet::colorNumeric()</code>.
zero_values_show	Logical indicating whether to show deployments with zero values. Default: <code>TRUE</code> .

zero_values_icon_url	Character with URL to icon for showing deployments with zero values. Default: a cross (multiply symbol) "https://img.icons8.com/ios-glyphs/30/000000/multiply.png".
zero_values_icon_size	A number to set the size of the icon to show deployments with zero values. Default: 10.
na_values_show	Logical indicating whether to show deployments with zero values. Notice that only feature "n_species" generates NA values. Default: TRUE.
na_values_icon_url	Character with URL to icon for showing deployments with NA values. Notice that only feature "n_species" generates NA values. Default: a red cross (multiply symbol) "https://img.icons8.com/ios-glyphs/30/FA5252/multiply.png".
na_values_icon_size	A number to set the size of the icon to show deployments with NA values. Notice that only feature "n_species" generates NA values. Default: 10.
relative_scale	Logical indicating whether to use a relative colour and radius scale (TRUE) or an absolute scale (FALSE). If absolute scale is used, specify a valid max_scale.
max_scale	Number indicating the max value used to map colour and radius.
radius_range	Vector of length 2 containing the lower and upper limit of the circle radius. The lower value is used for deployments with zero feature value, i.e. no observations, no identified species, zero RAI or zero effort. The upper value is used for the deployment(s) with the highest feature value (relative_scale = TRUE) or max_scale (relative_scale = FALSE). Default: c(10, 50).
datapkg	Deprecated. Use package instead.

Value

Leaflet map.

See Also

Check documentation about filter predicates: [pred\(\)](#), [pred_in\(\)](#), [pred_and\(\)](#), ...

Examples

```
## Not run:
# Show number of species
map_dep(
  mica,
  "n_species"
)

# Show number of observations (observations of unidentified species included
# if any)
map_dep(
  mica,
  "n_obs"
)
```

```
# Show number of observations of Anas platyrhynchos
map_dep(
  mica,
  "n_obs",
  species = "Anas platyrhynchos"
)

# Show number of observations of subadult individuals of Anas strepera
map_dep(
  mica,
  "n_obs",
  species = "Anas strepera",
  life_stage = "subadult"
)

# Show number of observations of female or unknown individuals of gadwall
map_dep(
  mica,
  "n_obs",
  species = "gadwall",
  sex = c("female", "unknown")
)

# Show number of individuals (individuals of unidentified species included if
# any)
map_dep(
  mica,
  "n_individuals"
)

# Same filters by life stage and sex as for number of observations apply
map_dep(
  mica,
  "n_individuals",
  species = "Anas strepera",
  sex = "female",
  life_stage = "adult"
)

# Show RAI
map_dep(
  mica,
  "rai",
  species = "Anas strepera"
)

# Same filters by life_stage and sex as for number of observations apply
map_dep(
  mica,
  "rai",
  species = "Anas strepera",
  sex = "female",
```

```
    life_stage = "adult"
  )

# Show RAI calculated by using number of detected individuals
map_dep(
  mica,
  "rai_individuals",
  species = "Anas strepera"
)

# Same filters by life stage and sex as for basic RAI apply
map_dep(
  mica,
  "rai_individuals",
  species = "Anas strepera",
  sex = "female",
  life_stage = "adult"
)

# Show effort (hours)
map_dep(
  mica,
  "effort"
)

# Show effort (days)
map_dep(
  mica,
  "effort",
  effort_unit = "day"
)

# Use viridis palette (viridis palettes)
map_dep(
  mica,
  "n_obs",
  palette = "viridis"
)

# Use "BuPu" colour palette (RColorBrewer palettes)
map_dep(
  mica,
  "n_obs",
  palette = "BuPu"
)

# Use a palette defined by colour names
map_dep(
  mica,
  "n_obs",
  palette = c("black", "blue", "white")
)

# Use a palette defined by hex colours
```

```

map_dep(
  mica,
  "n_obs",
  palette = c("#000000", "#0000FF", "#FFFFFF")
)

# Do not show deployments with zero values
map_dep(
  mica,
  "n_obs",
  life_stage = "subadult",
  zero_values_show = FALSE
)

# Use same icon but but a non default colour for zero values deployments,
# e.g. red (hex: E74C3C)
map_dep(
  mica,
  "n_obs",
  life_stage = "subadult",
  zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/multiply.png"
)

# ... or yellow (F1C40F)
map_dep(
  mica,
  "n_obs",
  life_stage = "subadult",
  zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/F1C40F/multiply.png"
)

# Use another icon via a different URL, e.g. the character Fry from Futurama
# in green (2ECC71)
map_dep(
  mica,
  "n_obs",
  life_stage = "subadult",
  zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/2ECC71/futurama-fry.png"
)

# Same behavior for the icon visualizing NA values (`n_species` feature)
unknown_species_vs_no_obs <- mica
unknown_species_vs_no_obs$data$observations <-
  unknown_species_vs_no_obs$data$observations %>%
  # a deployment has detected only unknown species
  filter(is.na(.data$scientificName) |
    .data$scientificName != "Homo sapiens") %>%
  # a deployment has no observations
  filter(deploymentID != "62c200a9-0e03-4495-bcd8-032944f6f5a1")
# create new map
map_dep(
  unknown_species_vs_no_obs,
  feature = "n_species",

```

```

zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/2ECC71/futurama-fry.png",
zero_values_icon_size = 60,
na_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/futurama-fry.png",
na_values_icon_size = 60
)

# Set size of the icon for zero values deployments
map_dep(
  mica,
  "n_obs",
  life_stage = "subadult",
  zero_values_icon_size = 30
)

# Disable cluster
map_dep(
  mica,
  "n_species",
  cluster = FALSE
)

# Show only number of observations and location name while hovering
map_dep(
  mica,
  "n_obs",
  hover_columns = c("locationName", "n")
)

# Use absolute scale for colours and radius
map_dep(mica,
  "n_species",
  relative_scale = FALSE,
  max_scale = 4
)

# Change max and min size circles
map_dep(
  mica,
  "n_obs",
  radius_range = c(40, 150)
)

## End(Not run)

```

mica

Sample of Camtrap DP formatted data

Description

A sample **Camera Trap Data Package** as read by `read_camtrap_dp()`. The source data are derived from the **Camtrap DP example dataset** and are saved in `inst/extdata/mica`.

Usage

```
mica
```

Format

An object of class `datapackage` (inherits from `list`) of length 16.

Details

A larger dataset is available in `inst/extdata/mica_zenodo_5590881`. It is derived from a dataset on [Zenodo](#), but excludes `media.csv`.

Source

<https://github.com/tdwg/camtrap-dp/tree/ad0278ef86ef518dacfb306c598dce97667cfb81/example>

See Also

Other sample data: [animal_positions](#), [dep_calib_models](#)

Examples

```
## Not run:  
# mica.rda was created with the code below.  
mica <- read_camtrap_dp(  
  system.file(  
    "extdata/mica",  
    "datapackage.json",  
    package = "camtraptor"  
  )  
)  
save(mica, file = "data/mica.rda")  
  
## End(Not run)
```

pred

Filter predicate

Description

Filter predicate

Usage

```

pred(arg, value)

pred_not(arg, value)

pred_gt(arg, value)

pred_gte(arg, value)

pred_lt(arg, value)

pred_lte(arg, value)

pred_in(arg, value)

pred_notin(arg, value)

pred_na(arg)

pred_notna(arg)

pred_and(...)

pred_or(...)

```

Arguments

arg	(character) The key for the predicate. See "Keys" below.
value	(various) The value for the predicate.
...	For <code>pred_or()</code> or <code>pred_and()</code> : one or more objects of class <code>filter_predicate</code> , created by any other <code>pred*</code> function.

Value

A predicate object. An object of class `predicate` is a list with the following elements:

- `arg`: A (list of) character with all arguments in the predicate(s).
- `value`: A (list of) character with all values in the predicate(s).
- `type`: A (list of) character with all predicate types, see section "predicate methods" here below.
- `expr`: A character: body of a filter expression.

Predicate methods and their equivalent types

`pred*` functions are named for the 'type' of operation they do, inspired by GBIF [occurrence predicates](#)

The following functions take one key and one value and are associated to the following types:

- `pred`: equals

- pred_not: notEquals
- pred_lt: lessThan
- pred_lte: lessThanOrEquals
- pred_gt: greaterThan
- pred_gte: greaterThanOrEquals
- pred_like: like (NOT IMPLEMENTED YET!)

The following function is only for geospatial queries, and only accepts a WKT string:

- pred_within: within (NOT IMPLEMENTED YET!)

The following functions are only for stating that you do (not) want a key to be NA, so only accepts one key:

- pred_na: isNA
- pred_notna: isNotNA

The following two functions accept multiple individual filter predicates, separating them by either "and" or "or":

- pred_and: and
- pred_or: or

The following function is special in that it accepts a single key but many values, stating that you want to search for all the listed values, e.g. one of the locations in: "B_ML_val 05_molenkreek", "B_ML_val 03_De Val" and "B_ML_val 06_Oostpolderkreek"

- pred_in: in
- pred_notin: notIn

What happens internally

Internally, the input to pred* functions turn into a character string, which forms the body of a filter expression. For example:

pred("tags", "boven de stroom") gives:

```
$arg
[1] "tags"
```

```
$value
[1] "boven de stroom"
```

```
$type
[1] "equals"
```

```
$expr
(tags == "boven de stroom")
```

pred_gt("latitude", 51.27) gives, (only expr element shown):

```
(latitude > 51.27)
```

pred_or() gives:

```
((tags == "boven de stroom") | (latitude > 51.28))
```

pred_or() gives:

```
((tags == "boven de stroom") & (latitude > 51.28))
```

Keys

Acceptable arguments to the key parameter are the column names of the data frame you are applying the filter predicates.

See Also

Other filter functions: [apply_filter_predicate\(\)](#)

Examples

```
# One arg one value predicates
pred("scientificName", "Anas platyrhynchos")
pred("tags", "boven de stroom")
pred_gt("latitude", 51.18)
pred_gte("latitude", 51.18)
pred_lt("longitude", 3.95)
pred_lte("longitude", 3.95)
pred_not("locationName", "B_DL_val 3_dikke boom")

# and/or predicates
pred_and(pred_lt("longitude", 3.59), pred_gt("latitude", 51.28))
pred_or(pred_gte("count", 2), pred("vernacular_name", "Norway Rat"))

# Use dates as argument
start_date <- as.Date("2020-06-03", format = "%Y-%m-%d")
end_date <- as.Date("2020-06-10", format = "%Y-%m-%d")
pred_or(pred_gte("start", start_date), pred_lte("end", end_date))

# Use datetimes (POSIXct) as argument
start_date <- lubridate::as_datetime("2020-06-03")
end_date <- lubridate::as_datetime("2020-06-10")
pred_or(pred_gte("start", start_date), pred_lte("end", end_date))

# One arg multiple values predicates
locations <- c("B_ML_val 03_De Val", "B_ML_val 05_molenkreek")
pred_in("location_name", locations)
pred_notin("location_name", locations)
start_dates <- lubridate::as_datetime(c("2020-06-03 20:10:18", "2020-06-03 20:04:33"))
pred_in("start", start_dates)
pred_notin("start", start_dates)
```

```
# One arg, no value predicates
pred_na("scientificName")
pred_notna("scientificName")
```

read_camtrap_dp	<i>Read a Camtrap DP</i>
-----------------	--------------------------

Description

Reads files from a **Camera Trap Data Package** into memory. All datetime information is automatically transformed to Coordinated Universal Time (UTC). Vernacular names found in the metadata (package\$taxonomic) are added to the observations data frame.

Usage

```
read_camtrap_dp(file = NULL, media = TRUE, path = lifecycle::deprecated())
```

Arguments

file	Path or URL to a datapackage.json file.
media	If TRUE (default), read media records into memory. If FALSE, ignore media file to speed up reading larger Camtrap DP packages.
path	Path to the directory containing the datapackage. Use file with path or URL to a datapackage.json file instead.

Value

List describing a Data Package (as returned by `frictionless::read_package()`) containing the original metadata, as well as a property data containing the data as three data frames:

1. deployments
2. media
3. observations

See Also

Other read functions: [read_wi\(\)](#)

Examples

```
## Not run:
# Read Camtrap DP package
camtrap_dp_file <- system.file(
  "extdata", "mica", "datapackage.json",
  package = "camtraptor"
)
muskrat_coypu <- read_camtrap_dp(camtrap_dp_file)
```

```
# Read Camtrap DP package and ignore media file
muskrat_coypu <- read_camtrap_dp(camtrap_dp_file, media = FALSE)

# If parsing issues while reading deployments, observations or media arise,
# use readr::problems()
camtrap_dp_file_with_issues <- system.file(
  "extdata",
  "mica_parsing_issues",
  "datapackage_for_parsing_issues.json",
  package = "camtraptor"
)
muskrat_coypu_with_issues <- read_camtrap_dp(camtrap_dp_file_with_issues, media = TRUE)
readr::problems(muskrat_coypu_with_issues$data$deployments)
readr::problems(muskrat_coypu_with_issues$data$observations)
readr::problems(muskrat_coypu_with_issues$data$media)

## End(Not run)
```

read_wi

Read a Wildlife Insights export

Description

Reads files from an unzipped **Wildlife Insights** export into memory. Data can be exported from Wildlife Insights as a **public** or **private** download. The function transforms data and metadata to a **Camera Trap Data Package** which can be written to file with `frictionless::write_package()`.

Usage

```
read_wi(directory = ".")
```

Arguments

directory Path to local directory to read files from. The function expects `projects.csv`, `deployments.csv`, `cameras.csv`, and `images.csv`.

Details

The function has only been tested on image-based projects.

Value

CSV (data) files written to disk.

See Also

Other read functions: `read_camtrap_dp()`

round_coordinates	<i>Round coordinates to generalize camera trap locations</i>
-------------------	--

Description

Rounds deployment coordinates to a certain number of digits to fuzzy/generalize camera trap locations. This function can be used before publishing data in order to protect sensitive species and/or prevent theft of active cameras.

Usage

```
round_coordinates(package, digits = 3)
```

Arguments

package	A Camtrap DP, as read by read_camtrap_dp() .
digits	Number of decimal places to round coordinates to (1, 2 or 3).

Value

package with rounded coordinates as well as updated `coordinateUncertainty` (in deployments) and `coordinatePrecision` (in metadata).

Details

Rounding coordinates is a recommended method to generalize sensitive biodiversity information (see [Section 4.2](#) in Chapman 2020). Choose a number of digits that aligns with the sensitivity of the data and notice the effect on precision and uncertainty. Publish the coordinates as is (i.e. do not use this function) if the data are not sensitive.

sensitivity	digits	coordinatePrecision	coordinateUncertainty
high	1	0.1	original uncertainty + 15691 m
medium	2	0.01	original uncertainty + 1570 m
low	3	0.001	original uncertainty + 157 m

For records with `coordinateUncertainty = NA` the function will assume the coordinates were obtained by GPS and use 30 m as original uncertainty, before adding uncertainty caused by rounding. The added uncertainty is the largest possible value caused by rounding (see [Table 3](#) in Chapman & Wieczorek 2020).

See Also

Other publication functions: [write_dwc\(\)](#), [write_eml\(\)](#)

Examples

```
# Round coordinates of example package to 3 digits
mica <- round_coordinates(mica, 3)

# coordinatePrecision is set in metadata
mica$coordinatePrecision

# coordinateUncertainty is set in data: original uncertainty (or 30) + 157 m
mica$data$deployments$coordinateUncertainty
```

write_dwc

Transform Camtrap DP data to Darwin Core

Description

Transforms data from a **Camera Trap Data Package** to **Darwin Core**. The resulting CSV files can be uploaded to an **IPT** for publication to GBIF. A meta.xml file is included as well. See `write_eml()` to create an `eml.xml` file.

Usage

```
write_dwc(package, directory = ".")
```

Arguments

package	A Camtrap DP, as read by <code>read_camtrap_dp()</code> .
directory	Path to local directory to write file(s) to. If NULL, then a list of data frames is returned instead, which can be useful for extending/adapting the Darwin Core mapping before writing with <code>readr::write_csv()</code> .

Value

CSV and meta.xml files written to disk or a list of data frames when `directory = NULL`.

Transformation details

Data are transformed into an **Occurrence core** and **Audubon Media Description extension**. This follows recommendations discussed and created by Peter Desmet, John Wiczorek, Lien Reyserhove, Ben Norton and others.

The following terms are set from the package metadata:

- **datasetName**: Title as provided in `package$title`.
- **datasetID**: Identifier as provided in `package$id`. Can be a DOI.
- **rightsHolder**: Rights holder as provided in `package$rightsHolder`.
- **collectionCode**: Platform name as provided in `package$platform$title`.
- **license**: License with scope data as provided in `package$licenses`.

- **rights** for media files: License with scope media as provided in package\$licenses.
- **dwc:dataGeneralizations**: "coordinates rounded to package\$coordinatePrecision degree".
- **coordinatePrecision**: package\$coordinatePrecision (e.g. 0.001).

Key features of the Darwin Core transformation:

- Deployments (of camera traps) are parent events, with observations (machine observations) as child events. No information about the parent event is provided other than its ID, meaning that data can be expressed in an Occurrence Core with one row per observation and parentEventID shared by all occurrences in a deployment.
- Sequence-based observations share an eventID per sequence, image-based observations share an eventID per image.
- The image(s) an observation is based on are provided in the [Audubon Media Description extension](#), with a foreign key to the observation.
- Excluded are records that document blank or unclassified media, vehicles and observations of humans.

See Also

Other publication functions: [round_coordinates\(\)](#), [write_eml\(\)](#)

write_eml

Transform Camtrap DP metadata to EML

Description

Transforms the metadata of a [Camera Trap Data Package](#) to an [EML](#) file that can be uploaded to a [GBIF IPT](#) for publication.

Usage

```
write_eml(
  package,
  directory = ".",
  title = package$title,
  description = package$description,
  creators = NULL,
  keywords = c("camera traps")
)
```

Arguments

package	A Camtrap DP, as read by read_camtrap_dp() .
directory	Path to local directory to write file to. If NULL, then the EML object is returned instead, which can be useful for extended/adapting the EML before writing with EML::write_eml() .

title	Dataset title.
description	Dataset description. Will be added after an automatically generated paragraph. Multiple paragraphs can be provided as a character vector.
creators	Dataset creators <ul style="list-style-type: none"> • If NULL then all <code>package\$contributors</code> will be added as creators, in the order as listed. • If e.g. <code>c("Emma Cartuyvels", "Jim Casaer", "...", "Peter Desmet")</code>, then Emma Cartuyvels, Jim Casaer and Peter Desmet will be set as first, second and last creators respectively, on the condition that their name (<code>title</code>) is present in <code>package\$contributors</code>. All other contributors will be inserted at "...", sorted on their last name.
keywords	Dataset keywords.

Value

`eml.xml` file written to disk or EML object when `directory = NULL`.

Transformation details

Metadata is derived from what is provided in `package` and in the function parameters. The following properties are set:

- **title**: Title as provided in `title` or `package$title`.
- **description**: Description as provided in `description` or `package$description`. The description is preceded by an automatically generated paragraph describing from which project and platform the dataset is derived, and to which extend coordinates are rounded (`package$coordinatePrecision`).
- **license**: License with scope data as provided in `package$licenses`.
- **creators**: Contributors (all roles) as provided in `package$contributors`, filtered/reordered based on `creators`.
- **contact**: First creator.
- **metadata provider**: First creator.
- **keywords**: Keywords as provided in `keywords`.
- **associated parties**: Organizations as provided in `package$organizations`.
- **geographic coverage**: Bounding box as provided in `package$spatial`.
- **taxonomic coverage**: Species (no other ranks) as provided in `package$taxonomic`.
- **temporal coverage**: Date range as provided in `package$temporal`.
- **project data**: Title, acronym as identifier, description, and sampling design as provided in `package$project`. The first creator is set as project personnel.
- **alternative identifier**: Identifier as provided in `package$id`. If this is a DOI, no new DOI will be created when publishing to GBIF.
- **external link**: URL of the project as provided in `package$project$path`.

To be set manually in the GBIF IPT: **type**, **subtype**, **update frequency** and **publishing organization**.

Not set: **sampling methods** and **citations**.

Not applicable: **collection data**.

See Also

Other publication functions: [round_coordinates\(\)](#), [write_dwc\(\)](#)

Index

- * **camtrapR-derived functions**
 - get_cam_op, 7
 - get_detection_history, 11
 - get_record_table, 23
 - * **datasets**
 - animal_positions, 2
 - dep_calib_models, 6
 - mica, 34
 - * **density estimation functions**
 - calc_animal_pos, 4
 - * **exploration functions**
 - get_custom_effort, 8
 - get_effort, 15
 - get_n_individuals, 16
 - get_n_obs, 18
 - get_n_species, 19
 - get_rai, 20
 - get_rai_individuals, 22
 - get_scientific_name, 26
 - get_species, 27
 - * **filter functions**
 - apply_filter_predicate, 3
 - pred, 35
 - * **publication functions**
 - round_coordinates, 41
 - write_dwc, 42
 - write_eml, 43
 - * **read functions**
 - read_camtrap_dp, 39
 - read_wi, 40
 - * **sample data**
 - animal_positions, 2
 - dep_calib_models, 6
 - mica, 34
 - * **validation functions**
 - check_species, 5
 - * **visualization functions**
 - map_dep, 28
- animal_positions, 2, 6, 35
- apply_filter_predicate, 3, 38
- calc_animal_pos, 4
- check_species, 5
- dep_calib_models, 3, 6, 35
- EML::write_eml(), 43
- frictionless::read_package(), 39
- frictionless::write_package(), 40
- get_cam_op, 7, 12, 25
- get_cam_op(), 12
- get_custom_effort, 8, 16–18, 20–22, 26, 27
- get_detection_history, 7, 11, 25
- get_effort, 9, 15, 17, 18, 20–22, 26, 27
- get_n_individuals, 9, 16, 16, 18, 20–22, 26, 27
- get_n_obs, 9, 16, 17, 18, 20–22, 26, 27
- get_n_species, 9, 16–18, 19, 21, 22, 26, 27
- get_rai, 9, 16–18, 20, 20, 22, 26, 27
- get_rai_individuals, 9, 16–18, 20, 21, 22, 26, 27
- get_record_table, 7, 12, 23
- get_record_table(), 12
- get_scientific_name, 9, 16–18, 20–22, 26, 27
- get_species, 9, 16–18, 20–22, 26, 27
- leaflet::colorNumeric(), 29
- map_dep, 28
- mica, 3, 6, 34
- pred, 3, 35
- pred(), 30
- pred_and(pred), 35
- pred_and(), 30
- pred_gt(pred), 35
- pred_gte(pred), 35

`pred_in(pred)`, 35
`pred_in()`, 30
`pred_lt(pred)`, 35
`pred_lte(pred)`, 35
`pred_na(pred)`, 35
`pred_not(pred)`, 35
`pred_notin(pred)`, 35
`pred_notna(pred)`, 35
`pred_or(pred)`, 35

`read_camtrap_dp`, 39, 40
`read_camtrap_dp()`, 34, 41–43
`read_wi`, 39, 40
`readr::write_csv()`, 42
`round_coordinates`, 41, 43, 45

`unmarked::unmarkedMultFrame()`, 12

`write_dwc`, 41, 42, 45
`write_eml`, 41, 43, 43