

Package: camtraptor (via r-universe)

October 7, 2026

Title Explore and Visualize Camera Trap Data Packages

Version 1.0.1

Date 2026-09-09

Description Explore and visualize Camera Trap Data Packages ('Camtrap DP', <<https://camtrap-dp.tdwg.org/>>). camtraptor offers a step-by-step workflow to read Camtrap DP files, filter data of interest, summarize information (e.g. number of observed species) and visualize this per deployment on an interactive map. You can also use it to transform data for analysis in 'camtrapR'.

License MIT + file LICENSE

URL <https://github.com/inbo/camtraptor>,
<https://inbo.github.io/camtraptor/>

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

Depends R (>= 4.1.0)

Imports activity, assertthat, camtrapdp, dplyr (>= 1.1.0), glue, htmltools, leaflet, lifecycle, lubridate, overlap, purrr, RColorBrewer, reshape2, rlang, snakecase, stringr, tidyr, utils

Suggests camtrapR, cli, covr, frictionless, knitr, rmarkdown, testthat (>= 3.0.0), unmarked, xml2

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

Roxygen list(markdown = TRUE)

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://inbo.r-universe.dev>

Date/Publication 2026-10-07 13:40:32 UTC

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

RemoteRef HEAD

RemoteSha 9f642800a4e7d39ca657a4506127dd2ff583dbbb

Contents

add_coordinates	3
animal_pos	3
animal_positions	4
calc_animal_pos	4
calculate_individual_radius_angle	6
calib_models	7
calibration_models	7
contributors	8
deployments	9
events	9
example_dataset	10
filter_deployments	11
filter_media	12
filter_observations	13
filter_out_timelapse	14
get_cam_op	15
get_custom_effort	17
get_detection_history	18
get_effort	22
get_n_individuals	23
get_n_obs	25
get_n_species	26
get_rai	27
get_rai_individuals	28
get_record_table	30
get_species	33
individuals	33
locations	34
map_dep	35
map_summary	41
media	47
mica	48
observations	49
read_camtrap_dp	49
read_camtrapdp	51
summarize_deployments	52
summarize_observations	54
taxa	57

add_coordinates	<i>Add deployment coordinates to observations</i>
-----------------	---

Description

This function adds deployment coordinates to observations based on deploymentID.

Usage

```
add_coordinates(x)
```

Arguments

`x` Camera trap data package object, as returned by `camtrapdp::read_camtrapdp()`.

Details

When the latitude and/or the longitude columns are already present in observations, a warning is issued and the original object is returned unchanged.

Value

Camera trap data package object, where observations is updated by appending two new columns: latitude and longitude

Examples

```
x <- example_dataset()

# Add coordinates to observations
add_coordinates(x) %>% observations()
```

animal_pos	<i>Deprecated datasets in camtraptor</i>
------------	--

Description

[Deprecated]

These datasets are deprecated:

- Use `example_dataset()` instead of mica.
- Use `animal_positions` instead of `animal_pos`. Note that the column `sequenceID` has been renamed to `eventID`.
- Use `calibration_models` instead of `calib_models`.

See Also

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

animal_positions	<i>Sample of animal position digitization data</i>
------------------	--

Description

A tibble data frame with the following columns:

- deploymentID
- eventID
- 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: [animal_pos](#), [calib_models](#), [calibration_models](#), [mica](#)

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

Description**[Deprecated]**

This function is deprecated. Use [calculate_individual_radius_angle\(\)](#) instead.

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 = "eventID",
  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 arguments deployment_id, 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_positions with additional columns:

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

Examples

```
calc_animal_pos(animal_positions, calibration_models)
```

```
calculate_individual_radius_angle
```

Calculate animal position

Description

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

Usage

```
calculate_individual_radius_angle(
  animal_positions,
  calibration_models,
  deployment_id = "deploymentID",
  event_id = "eventID",
  x = "x",
  y = "y",
  image_width = "imageWidth",
  image_height = "imageHeight"
)
```

Arguments

<code>animal_positions</code>	Data frame (tibble) of animal position digitization data. It must contain (at least) the columns defined in args <code>deployment_id</code> , <code>event_id</code> , <code>x</code> , <code>y</code> , <code>image_width</code> and <code>image_height</code> .
<code>calibration_models</code>	Named list of deployment calibration models or site calibration models (calibs objects), produced using <code>cal.site()</code> (not yet included in this package). The deployment names are used as names.
<code>deployment_id</code>	Column in <code>animal_positions</code> against which names of the elements can be matched to apply the right deployment calibration models. Default: "deploymentID".
<code>event_id</code>	Column in <code>animal_positions</code> containing the event ID the images belong to. Default: "eventID".
<code>x</code>	Column in <code>animal_positions</code> containing x pixel positions for each digitised point. Default: "x".
<code>y</code>	Column in <code>animal_positions</code> containing y pixel positions for each digitised point. Default: "y".
<code>image_width</code>	Column in <code>animal_positions</code> containing the pixel x dimension of each image. Default: "imageWidth". Notice that the pixel x dimension must be consistent for each deployment.
<code>image_height</code>	Column in <code>animal_positions</code> 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_positions` 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
calculate_individual_radius_angle(animal_positions, calibration_models)
```

calib_models	<i>Deprecated datasets in camtraptor</i>
--------------	--

Description**[Deprecated]**

These datasets are deprecated:

- Use `example_dataset()` instead of `mica`.
- Use `animal_positions` instead of `animal_pos`. Note that the column `sequenceID` has been renamed to `eventID`.
- Use `calibration_models` instead of `calib_models`.

See Also

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

calibration_models	<i>Sample of deployment calibration models</i>
--------------------	--

Description

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

Usage

```
calibration_models
```

Format

An object of class `calibs` of length 4.

See Also

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

contributors

Get or set contributors

Description

Re-export of `camtrapdp::contributors()`.

Usage

```
contributors(x)
```

```
contributors(x) <- value
```

Arguments

<code>x</code>	Camera Trap Data Package object, as returned by <code>read_camtrapdp()</code> .
<code>value</code>	A data frame to assign as contributors.

Value

A `tibble::tibble()` data frame with the contributors, containing the following columns (columns absent in `x$contributors` will be created):

- `title`
- `firstName`: if absent, this will be set to the first word in `title`, except if it is a single word or the role is `rightsHolder` or `publisher`.
- `lastName`: if absent, this will be set to the remaining words in `title`, with the same exceptions as `firstName`.
- `email`
- `path`
- `role`
- `organization`

See Also

Other accessor functions: `deployments()`, `events()`, `individuals()`, `locations()`, `media()`, `observations()`, `taxa()`

Examples

```
x <- example_dataset()
# Get contributors
contributors(x)

# Set contributors
contributors(x) <- head(contributors(x), 1)
```

deployments	<i>Get or set deployments</i>
-------------	-------------------------------

Description

Re-export of `camtrapdp::deployments()`.

Usage

```
deployments(x)

deployments(x) <- value
```

Arguments

x	Camera Trap Data Package object, as returned by <code>read_camtrapdp()</code> .
value	A data frame to assign as deployments.

Value

A `tibble::tibble()` data frame with deployments.

See Also

Other accessor functions: `contributors()`, `events()`, `individuals()`, `locations()`, `media()`, `observations()`, `taxa()`

Examples

```
x <- example_dataset()
# Get deployments
deployments(x)

# Set deployments (not recommended outside a function)
deployments(x) <- head(deployments(x), 1)
```

events	<i>Get events</i>
--------	-------------------

Description

Re-export of `camtrapdp::events()`.

Usage

```
events(x)
```

Arguments

`x` Camera Trap Data Package object, as returned by [read_camtrapdp\(\)](#).

Value

A [tibble::tibble\(\)](#) data frame with the events, containing the following columns:

- deploymentID
- eventID
- eventStart
- eventEnd

See Also

Other accessor functions: [contributors\(\)](#), [deployments\(\)](#), [individuals\(\)](#), [locations\(\)](#), [media\(\)](#), [observations\(\)](#), [taxa\(\)](#)

Examples

```
x <- example_dataset()
events(x)
```

`example_dataset`*Read the Camtrap DP example dataset*

Description

Re-export of [camtrapdp::example_dataset\(\)](#).

Usage

```
example_dataset()
```

Value

Camera Trap Data Package object.

See Also

Other read sample data functions: [mica](#)

Examples

```
example_dataset()
```

filter_deployments	<i>Filter deployments</i>
--------------------	---------------------------

Description

Re-export of `camtrapdp::filter_deployments()`.

Usage

```
filter_deployments(x, ...)
```

Arguments

x	Camera Trap Data Package object, as returned by <code>read_camtrapdp()</code> .
...	Filtering conditions, see <code>dplyr::filter()</code> .

Value

x filtered.

See Also

Other filter functions: `filter_media()`, `filter_observations()`, `filter_out_timelapse()`

Examples

```
x <- example_dataset()

# Filtering returns x, so pipe with deployments() to see the result
x |>
  filter_deployments(deploymentID == "62c200a9") |>
  deployments()

# Filtering on deployments also affects associated media and observations
x_filtered <- filter_deployments(x, deploymentID == "62c200a9")
media(x_filtered)
observations(x_filtered)

# Filtering on multiple conditions (combined with &)
x |>
  filter_deployments(latitude > 51.0, longitude > 5.0) |>
  deployments()

# Filtering on dates is easiest with lubridate
library(lubridate, warn.conflicts = FALSE)
x |>
  filter_deployments(lubridate::year(deploymentStart) == 2020) |>
  deployments()
```

```
x |>
  filter_deployments(
    deploymentStart >= lubridate::as_date("2020-06-19"),
    deploymentEnd <= lubridate::as_date("2020-08-30")
  ) |>
  deployments()
```

filter_media

Filter media

Description

Re-export of `camtrapdp::filter_media()`.

Usage

```
filter_media(x, ...)
```

Arguments

`x` Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`...` Filtering conditions, see `dplyr::filter()`.

Value

`x` filtered.

See Also

Other filter functions: `filter_deployments()`, `filter_observations()`, `filter_out_timelapse()`

Examples

```
x <- example_dataset()

# Filtering returns x, so pipe with media() to see the result
x |>
  filter_media(captureMethod == "timeLapse") |>
  media()

# Filtering on media also affects associated observations, but not deployments
x_filtered <- filter_media(x, favorite == TRUE)
observations(x_filtered)

# Filtering on multiple conditions (combined with &)
x |>
  filter_media(captureMethod == "activityDetection", filePublic == FALSE) |>
  media()
```

```
# Filtering on datetimes is easiest with lubridate
library(lubridate, warn.conflicts = FALSE)
x |>
  filter_media(lubridate::year(timestamp) == 2020) |>
  media()

x |>
  filter_media(
    timestamp >= lubridate::as_datetime("2020-08-02 05:01:00"),
    timestamp <= lubridate::as_datetime("2020-08-02 05:02:00")
  ) |>
  media()
```

filter_observations	<i>Filter observations</i>
---------------------	----------------------------

Description

Re-export of `camtrapdp::filter_observations()`.

Usage

```
filter_observations(x, ...)
```

Arguments

<code>x</code>	Camera Trap Data Package object, as returned by <code>read_camtrapdp()</code> .
<code>...</code>	Filtering conditions, see <code>dplyr::filter()</code> .

Value

`x` filtered.

See Also

Other filter functions: `filter_deployments()`, `filter_media()`, `filter_out_timelapse()`

Examples

```
x <- example_dataset()

# Filtering returns x, so pipe with observations() to see the result
x |>
  filter_observations(observationType == "animal") |>
  observations()

# Filtering on observations also affects associated media, but not deployments
x |>
  filter_observations(
```

```

      scientificName == "Vulpes vulpes",
      observationLevel == "event"
    ) |>
    media()
x |>
  filter_observations(
    scientificName == "Vulpes vulpes",
    observationLevel == "media"
  ) |>
  media()

# Filtering on multiple conditions (combined with &)
x |>
  filter_observations(
    deploymentID == "577b543a",
    scientificName %in% c("Martes foina", "Mustela putorius")
  ) |>
  observations()

# Filtering on datetimes is easiest with lubridate
library(lubridate, warn.conflicts = FALSE)
x |>
  filter_observations(lubridate::year(eventStart) == 2020) |>
  observations()

x |>
  filter_observations(
    eventStart >= lubridate::as_datetime("2020-06-19 22:00:00"),
    eventEnd <= lubridate::as_datetime("2020-06-19 22:10:00")
  ) |>
  observations()

```

`filter_out_timelapse` *Filter out timelapse observations*

Description

Subsets observations in a Camera Trap Data Package object, removing timelapse observations, i.e. observations where `captureMethod = timeLapse`. This function is a shortcut for `filter_observations(x, captureMethod != "timelapse")`.

Usage

```
filter_out_timelapse(x)
```

Arguments

`x` Camera trap data package object, as returned by `camtrapdp::read_camtrapdp()`.

Value

x filtered.

See Also

Other filter functions: [filter_deployments\(\)](#), [filter_media\(\)](#), [filter_observations\(\)](#)

Examples

```
x <- example_dataset()

# `x` doesn't contain timelapse observations, returned as is
filter_out_timelapse(x)

# Create a data package with timelapse observations
obs <- observations(x)
obs$captureMethod <- c(rep("timelapse", nrow(obs) - 1), "activityDetection")
observations(x) <- obs
# Filter out timelapse observations
filter_out_timelapse(x)
```

get_cam_op

Get the camera operation matrix

Description**[Superseded]**

This function is superseded because camtrapR supports reading Camera Trap Data Packages. Use [camtrapR::readCamtrapDP\(\)](#) and [camtrapR::cameraOperation\(\)](#) instead.

Creates a camera operation matrix. See [vignette\("camera-operation-matrix"\)](#) for more details on how to use this function.

Usage

```
get_cam_op(
  x,
  station_col = "locationName",
  camera_col = NULL,
  session_col = NULL,
  use_prefix = FALSE
)
```

Arguments

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

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 the 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. Rows are camera stations and columns are occasions (dates). Values indicate whether a camera was operational.

See Also

Other deprecated `camtrapR`-derived functions: `get_detection_history()`, `get_record_table()`

Examples

```
library(dplyr)
library(stringr)

x <- example_dataset()
get_cam_op(x)

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

# Specify column with session IDs
x_sessions <- x
deployments(x_sessions) <- deployments(x_sessions) %>%
  mutate(session = ifelse(
    str_starts(.data$locationName, "B_DL_"),
    "after2020",
    "before2020"
  ))

get_cam_op(x_sessions, session_col = "session")

# Specify column with camera IDs
```

```

x_cameras <- x_sessions
deployments(x_cameras) <- deployments(x_cameras) %>%
  mutate(cameraID = c(1, 2, 3, 4))
get_cam_op(x_cameras, camera_col = "cameraID")

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

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

```

get_custom_effort	<i>Get custom effort</i>
-------------------	--------------------------

Description

[Deprecated]

This function is deprecated. Use [summarize_deployments\(\)](#) instead.

Gets the effort for each deployment and a specific time interval such as day, week, month or year.

Usage

```

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

```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use filter_deployments() to filter on deployments.
start	[Deprecated] if not NULL . Not supported anymore. Use filter_deployments() to filter out deployments.
end	[Deprecated] if not NULL . Not supported anymore. Use filter_deployments() to filter out deployments.

group_by	Character, one of "day", "week", "month", "year". Default: NULL. See group_time_by argument in [summarize_deployments()].
unit	[Deprecated] The unit used to quantify the effort. Ignored as the effort is returned only as a duration object.

Value

A grouped tibble data frame with the following columns:

- group_by names, e.g. deploymentID, latitude, longitude and locationName.
- group_time_by name if provided, e.g. month. It contains the first date of the time interval, e.g. the first day of the month.
- effort_duration: A duration object (duration is a class from lubridate package). Duration is always recorded as a fixed number of seconds. See [lubridate::duration\(\)](#).

See Also

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

Examples

```
x <- example_dataset()

# Effort for each deployment over the entire duration of the project
get_custom_effort(x)

# Effort at monthly interval
get_custom_effort(x, group_by = "month")

# You can afterwards calculate the total effort over all deployments
library(dplyr)
get_custom_effort(x, group_by = "year") %>%
  filter(effort_duration > 0) %>%
  group_by(year) %>%
  summarise(
    deploymentIDs = list(deploymentID),
    ndep = n_distinct(deploymentID),
    effort_duration = sum(effort_duration)
  )
```

Description

[Superseded]

This function is superseded because `camtrapR` supports reading Camera Trap Data Packages. Use `camtrapR::readCamtrapDP()` and `camtrapR::detectionHistory()` instead.

Creates species detection matrices. See `vignette("detection-history-matrix")` for more details on how to use this function.

Usage

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

Arguments

<code>recordTable</code>	A record table created by <code>get_record_table()</code> .
<code>camOp</code>	A camera operation matrix created by <code>get_cam_op()</code> .
<code>species</code>	Character. The species name.
<code>output</code>	Character. The type of output. Choose one of: "binary", "n_observations", "n_individuals".
<code>occasionLength</code>	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

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 deprecated `camtrapR`-derived functions: [get_cam_op\(\)](#), [get_record_table\(\)](#)

Examples

```
library(dplyr)
library(stringr)

x <- example_dataset()
camOp <- get_cam_op(x)
recordTable <- get_record_table(x)
# More observations of the same species on the same day at the same station
# are left.
recordTable_multitple <- get_record_table(
  x,
  removeDuplicateRecords = FALSE
)

# Binary output
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "binary"
)

# Number of observations output: same as binary with default
# `get_record_table(x)`
get_detection_history(
  recordTable,
  camOp,
  species = "Anas platyrhynchos",
  output = "n_observations"
)
```

```
# Number of observations output: more than 1 if more than 1 observation of
# the species on the same day at the same station
get_detection_history(
  recordTable_multiple,
  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
x_sessions <- x
deployments(x_sessions) <- deployments(x_sessions) %>%
  mutate(session = ifelse(
    str_starts(.data$locationName, "B_DL_"),
    "after2020",
    "before2020"
  )
)
camOp_sessions <- get_cam_op(
  x_sessions,
  session_col = "session"
)
recordTable_sessions <- get_record_table(x_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

[Deprecated]

This function is deprecated. Use `summarize_deployments()` instead.

Gets the effort (deployment duration) per deployment.

Usage

```
get_effort(x, ..., unit = "hour")
```

Arguments

x	Camera trap data package object, as returned by <code>camtrapdp::read_camtrapdp()</code> .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use <code>filter_deployments()</code> to filter on deployments.
unit	[Deprecated] The unit used to quantify the effort. Ignored as the effort is returned only as a duration object.

Value

A grouped tibble data frame with the following columns:

- group_by names, e.g. deploymentID, latitude, longitude and locationName.
- group_time_by name if provided, e.g. month. It contains the first date of the time interval, e.g. the first day of the month.
- effort_duration: A duration object (duration is a class from lubridate package). Duration is always recorded as a fixed number of seconds. See `lubridate::duration()`.

See Also

Other deprecated exploration functions: `get_custom_effort()`, `get_n_individuals()`, `get_n_obs()`, `get_n_species()`, `get_rai()`, `get_rai_individuals()`

Examples

```
x <- example_dataset()
get_effort(x)
```

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

Description**[Deprecated]**

This function is deprecated. Use `summarize_observations()` instead.

Gets the number of individuals per deployment.

Usage

```
get_n_individuals(x, ..., species = "all", sex = NULL, life_stage = NULL)
```

Arguments

x	Camera trap data package object, as returned by <code>camtrapdp::read_camtrapdp()</code> .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use <code>filter_deployments()</code> to filter on deployments.
species	[Deprecated] Character with scientific names. Common names are not supported anymore as of camtrapdp 1.0.0. Please, check <code>filter_observations()</code> to know how to filter by scientificName. If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
sex	[Deprecated] 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. Please, check <code>filter_observations()</code> to know how to filter by sex.
life_stage	[Deprecated] 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. Please, check <code>filter_observations()</code> to know how to filter by lifeStage.

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 deprecated exploration functions: `get_custom_effort()`, `get_effort()`, `get_n_obs()`, `get_n_species()`, `get_rai()`, `get_rai_individuals()`

Examples

```
x <- example_dataset()

# Get sum of `individualCount` for each species
get_n_individuals(x)

# Get sum of `individualCount` of all species (unidentified included)
get_n_individuals(x, species = NULL)

# Get number of observations of some species only, e.g. Anas platyrhynchos
get_n_individuals(x, species = "Anas platyrhynchos")

# Specify sex
```

```
get_n_individuals(x, sex = "female")

# Specify life stage
get_n_individuals(x, life_stage = c("subadult", "adult"))
```

get_n_obs

*Get number of observations for each deployment***Description****[Deprecated]**

This function is deprecated. Use [summarize_observations\(\)](#) instead.

Gets the number of event-based observations per deployment.

Usage

```
get_n_obs(x, ..., species = "all", sex = NULL, life_stage = NULL)
```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use filter_deployments() to filter on deployments.
species	[Deprecated] Character with scientific names. Common names are not supported anymore as of camtrapdp 1.0.0. Please, check filter_observations() to know how to filter by scientificName. If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
sex	[Deprecated] 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. Please, check filter_observations() to know how to filter by sex.
life_stage	[Deprecated] 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. Please, check filter_observations() to know how to filter by lifeStage.

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

Examples

```
x <- example_dataset()

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

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

# Get number of observations of Anas platyrhynchos
get_n_obs(x, species = "Anas platyrhynchos")

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

# Specify life stage
get_n_obs(x, life_stage = c("subadult", "adult"))
```

get_n_species

Get number of identified species for each deployment

Description**[Deprecated]**

This function is deprecated. Use [summarize_observations\(\)](#) instead.

Usage

```
get_n_species(x, ...)
```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use filter_deployments() to filter on deployments.

Value

A tibble data frame with the following columns:

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

See Also

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

Examples

```
x <- example_dataset()
# Get number of species
get_n_species(x)
```

get_rai	<i>Get Relative Abundance Index (RAI)</i>
---------	---

Description

[Deprecated]

Usage

```
get_rai(x, ..., species = "all", sex = NULL, life_stage = NULL)
```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use filter_deployments() to filter on deployments.
species	[Deprecated] Character with scientific names. Common names are not supported anymore as of camtraptor 1.0.0. Please, check filter_observations() to know how to filter by scientificName. If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
sex	[Deprecated] 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. Please, check filter_observations() to know how to filter by sex.
life_stage	[Deprecated] 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. Please, check filter_observations() to know how to filter by lifeStage.

Details

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 event-based observations and effort is the effort duration in days.

It is deprecated as of camtraptor 1.0.0. Please use [summarize_observations\(\)](#) instead.

Value

A tibble data frame with the following columns:

- deploymentID: character vector with the deployment unique identifiers.
- scientificName: character vector with the scientific name.
- rai: numeric vector with the relative abundance index.

See Also

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

Examples

```
x <- example_dataset()

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

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

get_rai_individuals	<i>Get Relative Abundance Index (RAI) based on the sum of individual counts</i>
---------------------	---

Description

[Deprecated]

Usage

```
get_rai_individuals(x, ..., species = "all", sex = NULL, life_stage = NULL)
```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
...	[Deprecated] filter predicates for filtering on deployments are not supported anymore and an error is returned. Anything else throws a deprecation warning and will be ignored. Please use filter_deployments() to filter on deployments.
species	[Deprecated] Character with scientific names. Common names are not supported anymore as of camtrapdp 1.0.0. Please, check filter_observations() to know how to filter by scientificName. If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.

sex	[Deprecated] 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. Please, check filter_observations() to know how to filter by sex.
life_stage	[Deprecated] 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. Please, check filter_observations() to know how to filter by lifeStage.

Details

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 sum of individual counts related to event-based observations and effort is the effort duration in days.

It is deprecated as of camtraptor 1.0.0. Please use [summarize_observations\(\)](#) instead.

Value

A tibble data frame with the following columns:

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

See Also

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

Examples

```
x <- example_dataset()

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

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

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

Description

[Superseded]

This function is superseded because `camtrapR` supports reading Camera Trap Data Packages. Use `camtrapR::readCamtrapDP()` and `camtrapR::recordTable()` instead.

Creates a record table. All dates and times are expressed in UTC format. See `vignette("record-table")` for more details on how to use this function.

Usage

```
get_record_table(
  x,
  stationCol = "locationName",
  exclude = NULL,
  minDeltaTime = 0,
  deltaTimeComparedTo = NULL,
  removeDuplicateRecords = TRUE
)
```

Arguments

- | | |
|-------------------------------------|---|
| <code>x</code> | Camera trap data package object, as returned by <code>camtrapdp::read_camtrapdp()</code> . |
| <code>stationCol</code> | Character name of the column containing stations. Default: "locationName". |
| <code>exclude</code> | Character vector of scientific names to be excluded from the record table. Default: NULL. |
| <code>minDeltaTime</code> | Time difference between records of the same species at the same station to be considered independent (in minutes). Default: 0. |
| <code>deltaTimeComparedTo</code> | One of "lastIndependentRecord" or "lastRecord". For two records to be considered independent, the second one must be at least <code>minDeltaTime</code> minutes after the last independent record of the same species (<code>deltaTimeComparedTo = "lastIndependentRecord"</code>), or <code>minDeltaTime</code> minutes after the last record (<code>deltaTimeComparedTo = "lastRecord"</code>). If <code>minDeltaTime</code> is 0, <code>deltaTimeComparedTo</code> must be NULL (default). |
| <code>removeDuplicateRecords</code> | Logical. If there are several records of the same species, but e.g. different sex or lifeStage, at the same station at exactly the same time, show only one? Default: TRUE. Duplicates are removed by keeping only the first observation in the observation table. |

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 observed individuals (renamed from `count` in the observations table).
- **DateTimeOriginal:** Datetime object, as found in column `eventStart` 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) [doi:10.1111/j.14697998.2011.00864.x](https://doi.org/10.1111/j.14697998.2011.00864.x).

See Also

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

Examples

```
library(lubridate)

x <- example_dataset()
get_record_table(x)

# Create a new Camera Trap Data Package with dependent observations only for
# demonstration.
```

```

obs <- observations(x)
obs[obs$observationID == "9e191d10",]$scientificName <- "Martes foina"
x_dep <- x
observations(x_dep) <- obs

# Set a minDeltaTime of 100 minutes from last record
get_record_table(
  x_dep,
  minDeltaTime = 100,
  deltaTimeComparedTo = "lastRecord"
)

# Differences can occur between `deltaTimeCoparedTo` = "lastRecord" and
# "lastIndependentRecord"
obs <- observations(x)
obs[obs$eventID == "02ae9f43", "eventStart"] <-
  as_datetime("2020-08-02 05:10:20")

med <- media(x)
rows_to_update <- which(med$eventID == "02ae9f43")
med[rows_to_update, "timestamp"] <- as_datetime("2020-08-02 05:10:20")

x_modified <- x
observations(x_modified) <- obs
media(x_modified) <- med

rec_last_indep <- get_record_table(
  x_modified,
  minDeltaTime = 10,
  deltaTimeComparedTo = "lastIndependentRecord"
)

rec_last <- get_record_table(
  x_modified,
  minDeltaTime = 10,
  deltaTimeComparedTo = "lastRecord"
)

# Exclude observations of Anas platyrhynchos.
get_record_table(x, exclude = "Anas platyrhynchos")

# Specify column to pass station names
get_record_table(x, stationCol = "locationID")

# Include "duplicates", i.e. records of same species at same time, but
# different attributes, such as life stage or sex.
get_record_table(
  x,
  removeDuplicateRecords = FALSE
)

```

`get_species`*Get species*

Description**[Deprecated]**

It is deprecated as of camtraptor 1.0.0. Please use [taxa\(\)](#) instead.

Usage

```
get_species(x)
```

Arguments

`x` Camera trap data package object, as returned by [camtrapdp::read_camtrapdp\(\)](#).

Value

A tibble data frame with taxonomic information.

Examples

```
x <- example_dataset()
get_species(x)
```

`individuals`*Get individuals*

Description

Re-export of [camtrapdp::individuals\(\)](#).

Usage

```
individuals(x)
```

Arguments

`x` Camera Trap Data Package object, as returned by [read_camtrapdp\(\)](#).

Value

A `tibble::tibble()` data frame with the individuals that have an individualID, containing the following columns:

- individualID
- scientificName
- lifeStage
- sex

See Also

Other accessor functions: `contributors()`, `deployments()`, `events()`, `locations()`, `media()`, `observations()`, `taxa()`

Examples

```
x <- example_dataset()
individuals(x)
```

locations

*Get locations***Description**

Re-export of `camtrapdp::locations()`.

Usage

```
locations(x)
```

Arguments

x Camera Trap Data Package object, as returned by `read_camtrapdp()`.

Value

A `tibble::tibble()` data frame with the locations, containing the following columns:

- locationID
- locationName
- latitude
- longitude
- coordinateUncertainty

See Also

Other accessor functions: [contributors\(\)](#), [deployments\(\)](#), [events\(\)](#), [individuals\(\)](#), [media\(\)](#), [observations\(\)](#), [taxa\(\)](#)

Examples

```
x <- example_dataset()
locations(x)
```

map_dep

*Visualize deployments features***Description****[Deprecated]**

This function is deprecated. Use [map_summary\(\)](#) instead.

Usage

```
map_dep(
  x,
  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)
)
```

Arguments

x	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
feature	Feature to visualize. One of the columns generated by summarize_deployments() or summarize_observations() not used for grouping. Possible values present in output of summarize_observations() are:

- `n_scientificName`: Number of detected species.
- `n_events`: Number of events observed.
- `n_observations`: Number of observations.
- `sum_count`: Number of individuals observed.
- `rai_observations`: RAI calculated using the number of observations (`n_observations`).
- `rai_count`: RAI calculated using the number of observed individuals (`sum_count`).

Possible values present in output of `summarize_deployments()` are:

- `effort_duration`: Deployment effort duration.

<code>species</code>	[Deprecated] Character with scientific names. Common names are not supported anymore as of <code>camtrapr</code> 1.0.0. Please, check <code>filter_observations()</code> to know how to filter by <code>scientificName</code> . If "all" (default) all scientific names are automatically selected. If NULL all observations of all species are taken into account.
<code>sex</code>	[Deprecated] 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. Please, check <code>filter_observations()</code> to know how to filter by sex.
<code>life_stage</code>	[Deprecated] 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. Please, check <code>filter_observations()</code> to know how to filter by <code>lifeStage</code> .
<code>effort_unit</code>	Time unit to use while visualizing deployment effort duration. Ignored if feature is not "effort_duration". One of: • second • minute • hour • day • month • year If NULL (default), the effort is returned in hours.
<code>cluster</code>	Logical value indicating whether using the cluster option while visualizing maps. Default: TRUE.
<code>hover_columns</code>	Character vector of the columns of <code>df</code> to show on mouse hover. Use NULL to disable hovering. Default: all columns in <code>df</code> used for grouping and the feature to visualize.
<code>palette</code>	The palette name or the color function that values will be mapped to. Default: "inferno". Typically one of the following: • A character vector of RGB or named colors. Examples: <code>c("#000000", "#0000FF", "#FFFFFF")</code>, <code>t</code> • The full name of a <code>RColorBrewer</code> palette, e.g. "BuPu" or "Greens", or viridis palette: "viridis", "magma", "inferno" or "plasma". For more options, see parameter <code>palette</code> of <code>leaflet::colorNumeric()</code> .
<code>zero_values_show</code>	Logical indicating whether to show deployments with zero values. Default: TRUE. See details.

zero_values_icon_url	Character with URL to icon for showing deployments with zero values. Use NULL if zero_values_show is FALSE. Default: a cross (multiply symbol) "https://img.icons8.com/ios-glyphs/30/000000/multiply.png". See details.
zero_values_icon_size	A number to set the size of the icon to show groups (e.g. deployments) with zero values. Use NULL if zero_values_show is FALSE. Default: a cross (multiply symbol) "https://img.icons8.com/ios-glyphs/30/000000/multiply.png". See details.
na_values_show	Logical indicating whether to show groups (e.g. deployments) with no observations when feature is "n_scientificName". Default: TRUE. See details.
na_values_icon_url	Character with URL to icon for showing groups (e.g. deployments) with no observations. Used only if feature is "n_scientificName", ignored otherwise. Use NULL if na_values_show is FALSE. 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 for showing groups (e.g. deployments) with no observations. Used only if feature is "n_scientificName", ignored otherwise. Use NULL if na_values_show is FALSE. Default: 10.
relative_scale	Logical indicating whether to use a relative color 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 color 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, e.g. 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).

Details

Deployments with zero values are shown only if they are present in the summary, df. If you want to show deployments with zero values not present in the summary, be sure to have used `extend = TRUE` in `summarize_observations()`. See the examples.

Notice also that the argument `species` is not present anymore. If you want to visualize a feature only for a specific group of species, please filter the Camera Trap Data Package using `filter_observations()` before using this function. See the examples.

Value

Leaflet map.

Examples

```
x <- example_dataset()
# Filter a data package to get only the data of Anas platyrhynchos
x_anas_p <- x %>%
  filter_observations(scientificName == "Anas platyrhynchos")
```

```

# Show number of species
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_scientificName")

# Show number of species: show deployments with no
# observations (NA icon and size as defaults)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary("n_scientificName")

# Show total number of observations (all species)
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_observations")

# Show number of observations of Anas platyrhynchos - show deployments with
# no observations (zero icon and size as defaults)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary("n_observations")

# Show number of observations of Anas platyrhynchos without extending the
# summary: deployments with no observations are not shown
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_observations")

# Same result as above if you extend the summary but use
# `zero_values_show` = `FALSE`
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary(
    "n_observations",
    zero_values_show = FALSE,
    zero_values_icon_url = NULL,
    zero_values_icon_size = NULL
  )

```

```

    )

# You can eventually filter the species after summarizing,
# although it could not be the most efficient way for large Camera Trap Data
# Packages
library(dplyr)
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude", "scientificName")
  ) %>%
  filter(scientificName == "Anas platyrhynchos") %>%
  map_summary("n_scientificName")

# Show number of individuals
x_anas_p %>%
  summarize_observations() %>%
  map_summary("sum_count")

# Show RAI
x_anas_p %>%
  summarize_observations() %>%
  map_summary("rai_observations")

# Show RAI (individual counts)
x_anas_p %>%
  summarize_observations() %>%
  map_summary("rai_count")

# Show effort (hours)
x_anas_p %>%
  summarize_deployments() %>%
  map_summary("effort_duration")

# Show effort (days)
x %>%
  summarize_deployments() %>%
  map_summary("effort_duration", effort_unit = "day")

# Show effort per month
x %>%
  summarize_deployments(group_time_by = "month") %>%
  map_summary("effort_duration", effort_unit = "day")

# Use viridis palette
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = "viridis")

# Use a palette defined by color names
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = c("black", "blue", "white"))

```

```

# Use a palette defined by hex colors
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = c("#000000", "#0000FF", "#FFFFFF"))

# Use same icon but a non default color for zero values deployments,
# E.g. red (hex: E74C3C)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary(
    "n_observations",
    zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/multiply.png"
  )

# Use another icon via a different URL, e.g. the character Fry from Futurama
# in green (2ECC71)
x_anas_p %>%
  summarize_observations(extend = TRUE) %>%
  map_summary(
    "n_observations",
    zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/2ECC71/futurama-fry.png"
  )

# Set size of the icon for zero values deployments
x %>%
  filter_observations(scientificName == "Anas platyrhynchos") %>%
  summarize_observations(extend = TRUE) %>%
  map_summary("n_observations", zero_values_icon_size = 30)

# Use another icon url/size for visualizing groups with no observations
# (NA, only for `n_scientificName` feature)
x %>%
  filter_observations(deploymentID != "00a2c20d") %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary(
    feature = "n_scientificName",
    na_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/futurama-fry.png",
    na_values_icon_size = 60
  )

# Disable cluster
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_observations", cluster = FALSE)

# Show only number of observations and location name while hovering

```

```

x %>%
  summarize_observations(
    group_by = c("deploymentID", "locationName", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_observations",
    hover_columns = c("locationName", "n_observations")
  )

# Use absolute scale for colors and radius
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_scientificName",
    relative_scale = FALSE,
    max_scale = 4
  )

# Change max and min size circles
x %>%
  summarize_observations(
    group_by = c("deploymentID", "locationName", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_observations",
    radius_range = c(40, 150)
  )

```

map_summary

Visualize deployments features

Description

This function visualizes deployments features such as number of detected species, number of observations and RAI (Relative Abundance Index) on a dynamic map. The circle size and color are proportional to the mapped feature.

Usage

```

map_summary(
  df,
  feature,
  effort_unit = NULL,
  cluster = TRUE,
  hover_columns = c(dplyr::group_vars(df), feature),
  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)
)

```

Arguments

df	Grouped data frame generated by <code>summarize_deployments()</code> or <code>summarize_observations()</code> or eventually a subset of it. If columns latitude and longitude are not present in df, no map can be generated and an error is returned.
feature	Feature to visualize. One of the columns generated by <code>summarize_deployments()</code> or <code>summarize_observations()</code> not used for grouping. Possible values present in output of <code>summarize_observations()</code> are: • <code>n_scientificName</code>: Number of detected species. • <code>n_events</code>: Number of events observed. • <code>n_observations</code>: Number of observations. • <code>sum_count</code>: Number of individuals observed. • <code>rai_observations</code>: RAI calculated using the number of observations (<code>n_observations</code>). • <code>rai_count</code>: RAI calculated using the number of observed individuals (<code>sum_count</code>). Possible values present in output of <code>summarize_deployments()</code> are: • <code>effort_duration</code>: Deployment effort duration.
effort_unit	Time unit to use while visualizing deployment effort duration. Ignored if feature is not "effort_duration". One of: • <code>second</code> • <code>minute</code> • <code>hour</code> • <code>day</code> • <code>month</code> • <code>year</code> If NULL (default), the effort is returned in hours.
cluster	Logical value indicating whether using the cluster option while visualizing maps. Default: TRUE.
hover_columns	Character vector of the columns of df to show on mouse hover. Use NULL to disable hovering. Default: all columns in df used for grouping and the feature to visualize.
palette	The palette name or the color function that values will be mapped to. Default: "inferno". Typically one of the following: • A character vector of RGB or named colors. Examples: <code>c("#000000", "#0000FF", "#FFFFFF")</code>, <code>t</code>

- The full name of a RColorBrewer palette, e.g. "BuPu" or "Greens", or viridis palette: "viridis", "magma", "inferno" or "plasma".

For more options, see parameter palette of `leaflet::colorNumeric()`.

`zero_values_show`

Logical indicating whether to show deployments with zero values. Default: TRUE. See details.

`zero_values_icon_url`

Character with URL to icon for showing deployments with zero values. Use NULL if `zero_values_show` is FALSE. Default: a cross (multiply symbol) "<https://img.icons8.com/ios-glyphs/30/000000/multiply.png>". See details.

`zero_values_icon_size`

A number to set the size of the icon to show groups (e.g. deployments) with zero values. Use NULL if `zero_values_show` is FALSE. Default: a cross (multiply symbol) "<https://img.icons8.com/ios-glyphs/30/000000/multiply.png>". See details.

`na_values_show` Logical indicating whether to show groups (e.g. deployments) with no observations when feature is "n_scientificName". Default: TRUE. See details.

`na_values_icon_url`

Character with URL to icon for showing groups (e.g. deployments) with no observations. Used only if feature is "n_scientificName", ignored otherwise. Use NULL if `na_values_show` is FALSE. 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 for showing groups (e.g. deployments) with no observations. Used only if feature is "n_scientificName", ignored otherwise. Use NULL if `na_values_show` is FALSE. Default: 10.

`relative_scale` Logical indicating whether to use a relative color 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 color 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, e.g. 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)`.

Details

Deployments with zero values are shown only if they are present in the summary, `df`. If you want to show deployments with zero values not present in the summary, be sure to have used `extend = TRUE` in `summarize_observations()`. See the examples.

Notice also that the argument `species` is not present anymore. If you want to visualize a feature only for a specific group of species, please filter the Camera Trap Data Package using `filter_observations()` before using this function. See the examples.

Value

Leaflet map.

Examples

```
x <- example_dataset()
# Filter a data package to get only the data of Anas platyrhynchos
x_anas_p <- x %>%
  filter_observations(scientificName == "Anas platyrhynchos")

# Show number of species
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_scientificName")

# Show number of species: show deployments with no
# observations (NA icon and size as defaults)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary("n_scientificName")

# Show total number of observations (all species)
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_observations")

# Show number of observations of Anas platyrhynchos - show deployments with
# no observations (zero icon and size as defaults)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary("n_observations")

# Show number of observations of Anas platyrhynchos without extending the
# summary: deployments with no observations are not shown
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary("n_observations")

# Same result as above if you extend the summary but use
# `zero_values_show` = `FALSE`
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
```

```

    ) %>%
    map_summary(
      "n_observations",
      zero_values_show = FALSE,
      zero_values_icon_url = NULL,
      zero_values_icon_size = NULL
    )

# You can eventually filter the species after summarizing,
# although it could not be the most efficient way for large Camera Trap Data
# Packages
library(dplyr)
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude", "scientificName")
  ) %>%
  filter(scientificName == "Anas platyrhynchos") %>%
  map_summary("n_scientificName")

# Show number of individuals
x_anas_p %>%
  summarize_observations() %>%
  map_summary("sum_count")

# Show RAI
x_anas_p %>%
  summarize_observations() %>%
  map_summary("rai_observations")

# Show RAI (individual counts)
x_anas_p %>%
  summarize_observations() %>%
  map_summary("rai_count")

# Show effort (hours)
x_anas_p %>%
  summarize_deployments() %>%
  map_summary("effort_duration")

# Show effort (days)
x %>%
  summarize_deployments() %>%
  map_summary("effort_duration", effort_unit = "day")

# Show effort per month
x %>%
  summarize_deployments(group_time_by = "month") %>%
  map_summary("effort_duration", effort_unit = "day")

# Use viridis palette
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = "viridis")

```

```

# Use a palette defined by color names
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = c("black", "blue", "white"))

# Use a palette defined by hex colors
x_anas_p %>%
  summarize_observations() %>%
  map_summary("n_observations", palette = c("#000000", "#0000FF", "#FFFFFF"))

# Use same icon but a non default color for zero values deployments,
# E.g. red (hex: E74C3C)
x_anas_p %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude"),
    extend = TRUE
  ) %>%
  map_summary(
    "n_observations",
    zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/multiply.png"
  )

# Use another icon via a different URL, e.g. the character Fry from Futurama
# in green (2ECC71)
x_anas_p %>%
  summarize_observations(extend = TRUE) %>%
  map_summary(
    "n_observations",
    zero_values_icon_url = "https://img.icons8.com/ios-glyphs/30/2ECC71/futurama-fry.png"
  )

# Set size of the icon for zero values deployments
x %>%
  filter_observations(scientificName == "Anas platyrhynchos") %>%
  summarize_observations(extend = TRUE) %>%
  map_summary("n_observations", zero_values_icon_size = 30)

# Use another icon url/size for visualizing groups with no observations
# (NA, only for `n_scientificName` feature)
x %>%
  filter_observations(deploymentID != "00a2c20d") %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary(
    feature = "n_scientificName",
    na_values_icon_url = "https://img.icons8.com/ios-glyphs/30/E74C3C/futurama-fry.png",
    na_values_icon_size = 60
  )

# Disable cluster
x %>%

```

```

summarize_observations(
  group_by = c("deploymentID", "latitude", "longitude")
) %>%
map_summary("n_observations", cluster = FALSE)

# Show only number of observations and location name while hovering
x %>%
  summarize_observations(
    group_by = c("deploymentID", "locationName", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_observations",
    hover_columns = c("locationName", "n_observations")
  )

# Use absolute scale for colors and radius
x %>%
  summarize_observations(
    group_by = c("deploymentID", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_scientificName",
    relative_scale = FALSE,
    max_scale = 4
  )

# Change max and min size circles
x %>%
  summarize_observations(
    group_by = c("deploymentID", "locationName", "latitude", "longitude")
  ) %>%
  map_summary(
    "n_observations",
    radius_range = c(40, 150)
  )

```

media

Get or set media

Description

Re-export of `camtrapdp::media()`.

Usage

```
media(x)
```

```
media(x) <- value
```

Arguments

`x` Camera Trap Data Package object, as returned by [read_camtrapdp\(\)](#).
`value` A data frame to assign as media.

Value

A [tibble::tibble\(\)](#) data frame with media.

See Also

Other accessor functions: [contributors\(\)](#), [deployments\(\)](#), [events\(\)](#), [individuals\(\)](#), [locations\(\)](#), [observations\(\)](#), [taxa\(\)](#)

Examples

```
x <- example_dataset()
# Get media
media(x)

# Set media (not recommended outside a function)
media(x) <- head(media(x), 1)
```

mica

Deprecated datasets in camtraptor

Description**[Deprecated]**

These datasets are deprecated:

- Use [example_dataset\(\)](#) instead of mica.
- Use `animal_positions` instead of `animal_pos`. Note that the column `sequenceID` has been renamed to `eventID`.
- Use `calibration_models` instead of `calib_models`.

See Also

Other sample data: [animal_pos](#), [animal_positions](#), [calib_models](#), [calibration_models](#)

Other read sample data functions: [example_dataset\(\)](#)

observations	<i>Get or set observations</i>
--------------	--------------------------------

Description

Re-export of `camtrapdp::observations()`.

Usage

```
observations(x)
```

```
observations(x) <- value
```

Arguments

x	Camera Trap Data Package object, as returned by <code>read_camtrapdp()</code> .
value	A data frame to assign as observations.

Value

A `tibble::tibble()` data frame with observations.

See Also

Other accessor functions: `contributors()`, `deployments()`, `events()`, `individuals()`, `locations()`, `media()`, `taxa()`

Examples

```
x <- example_dataset()
# Get the observations
observations(x)

# Set observations (not recommended outside a function)
observations(x) <- head(observations(x), 1)
```

read_camtrap_dp	<i>Read a Camera Trap Data Package</i>
-----------------	--

Description**[Deprecated]**

This function is deprecated. Use `read_camtrapdp()` instead.

Reads files from a **Camera Trap Data Package (Camtrap DP)** into memory.

Usage

```
read_camtrap_dp(file)
```

Arguments

file	Path or URL to a datapackage.json file.
------	---

Value

Camera Trap Data Package object.

Older versions

The `read_camtrapdp()` function supports older versions of Camtrap DP and will automatically **upgrade** such datasets to the latest version of the standard. It currently supports versions 1.0, 1.0.1 and 1.0.2 (latest).

Events

Observations can contain classifications at two levels:

- **Media-based** observations (`observationLevel = "media"`) are based on a single media file and are directly linked to it via `mediaID`.
- **Event-based** observations (`observationLevel = "event"`) are based on an event, defined as a combination of `eventID`, `eventStart` and `eventEnd`. This event can consist of one or more media files, but is not directly linked to these.

The `read_camtrapdp()` function **will automatically assign eventIDs to media**, using `media.deploymentID = observations.deploymentID` and `observations.eventStart <= media.timestamp <= observations.eventEnd`. Note that this can result in media being linked to multiple events (and thus being duplicated), for example when events and sub-events were defined.

Taxonomic information

Camtrap DP metadata has a `taxonomic` property that can contain extra information for each `scientificName` found in observations. Such information can include higher taxonomy (family, order, etc.) and vernacular names in multiple languages.

The `read_camtrapdp()` function **will automatically include this taxonomic information in observations**, as extra columns starting with `taxon..` It will then update the taxonomic scope in the metadata to the unique `taxa()` found in the data.

Spatial/temporal coverage

Camtrap DP metadata has a `spatial` and `temporal` property that contains the spatial and temporal coverage of the package respectively.

The `read_camtrapdp()` function **will automatically update (or create) the spatial and temporal scopes** in the metadata based on the data. It also does this for the taxonomic scope (see higher).

Additional resources

A Camtrap DP can contain Data Resources not described by the standard. Those are listed with the tables supported by the standard (i.e. deployments, media, observations) in the resources property.

The `read_camtrapdp()` function will **ignore these additional resources** and only read the tables described by the standard. Additional resources can be read with `frictionless::read_resource()` if they are tabular.

Examples

```
file <- "https://raw.githubusercontent.com/tdwg/camtrap-dp/1.0.2/example/datapackage.json"
x <- read_camtrapdp(file)
x
```

read_camtrapdp

Read a Camera Trap Data Package

Description

Re-export of `camtrapdp::read_camtrapdp()`.

Usage

```
read_camtrapdp(file)
```

Arguments

`file` Path or URL to a `datapackage.json` file.

Value

A Camera Trap Data Package object.

Older versions

The `read_camtrapdp()` function supports older versions of Camtrap DP and will automatically **upgrade** such datasets to the latest version of the standard. It currently supports versions 1.0, 1.0.1 and 1.0.2 (latest).

Events

Observations can contain classifications at two levels:

- **Media-based** observations (`observationLevel = "media"`) are based on a single media file and are directly linked to it via `mediaID`.
- **Event-based** observations (`observationLevel = "event"`) are based on an event, defined as a combination of `eventID`, `eventStart` and `eventEnd`. This event can consist of one or more media files, but is not directly linked to these.

The `read_camtrapdp()` function **will automatically assign eventIDs to media**, using `media.deploymentID = observations.deploymentID` and `observations.eventStart <= media.timestamp <= observations.eventEnd`. Note that this can result in media being linked to multiple events (and thus being duplicated), for example when events and sub-events were defined.

Taxonomic information

Camtrap DP metadata has a taxonomic property that can contain extra information for each `scientificName` found in observations. Such information can include higher taxonomy (family, order, etc.) and vernacular names in multiple languages.

The `read_camtrapdp()` function **will automatically include this taxonomic information in observations**, as extra columns starting with `taxon..` It will then update the taxonomic scope in the metadata to the unique `taxa()` found in the data.

Spatial/temporal coverage

Camtrap DP metadata has a `spatial` and `temporal` property that contains the spatial and temporal coverage of the package respectively.

The `read_camtrapdp()` function **will automatically update (or create) the spatial and temporal scopes** in the metadata based on the data. It also does this for the taxonomic scope (see higher).

Additional resources

A Camtrap DP can contain Data Resources not described by the standard. Those are listed with the tables supported by the standard (i.e. deployments, media, observations) in the `resources` property.

The `read_camtrapdp()` function will **ignore these additional resources** and only read the tables described by the standard. Additional resources can be read with `frictionless::read_resource()` if they are tabular.

Examples

```
file <- "https://raw.githubusercontent.com/tdwg/camtrap-dp/1.0.2/example/datapackage.json"
x <- read_camtrapdp(file)
x
```

`summarize_deployments` *Summarize deployments information*

Description

Summarizes deployments information, more specifically the duration effort.

Usage

```
summarize_deployments(
  x,
  group_by = c("deploymentID", "latitude", "longitude"),
  group_time_by = NULL
)

summarise_deployments(
  x,
  group_by = c("deploymentID", "latitude", "longitude"),
  group_time_by = NULL
)
```

Arguments

<code>x</code>	Camera trap data package object, as returned by camtrapdp::read_camtrapdp() .
<code>group_by</code>	Character vector with the names of the columns in deployments. At the moment you can choose one or many columns among: <code>c("deploymentID", "latitude", "longitude", "locationID", "locationName", "deploymentStart", "deploymentEnd", "deploymentTags")</code> . Default: <code>c("deploymentID", "latitude", "longitude")</code> .
<code>group_time_by</code>	Character, one of "day", "week", "month", "year". The effort is calculated at the interval rate defined in <code>group_time_by</code> . Default: NULL, no grouping, i.e. the entire duration of the deployment is taken into account as a whole.

Details

`summarize_deployments()` and `summarise_deployments()` are synonyms.

Value

A grouped tibble data frame with the following columns:

- `group_by` names, e.g. `deploymentID`, `latitude`, `longitude` and `locationName`.
- `group_time_by` name if provided, e.g. `month`. It contains the first date of the time interval, e.g. the first day of the month.
- `effort_duration`: A duration object (duration is a class from lubridate package). Duration is always recorded as a fixed number of seconds. See [lubridate::duration\(\)](#).

See Also

Other exploration functions: [summarize_observations\(\)](#)

Examples

```
x <- example_dataset()

# Return effort using default `group_by` and no time grouping
summarize_deployments(x)
```

```

# Return effort using default `group_by` and grouping by year
summarize_deployments(x, group_time_by = "year")

# Return effort specifying grouping columns, e.g. `deploymentID` and
# `locationName` and grouping by day
summarize_deployments(
  x,
  group_by = c("deploymentID", "locationName"),
  group_time_by = "day"
)

# Afterwards, you can calculate the total effort over all deployments. You
# can also show other information, e.g. the (number of) deployments and
# locations.
library(dplyr)
summarize_deployments(
  x,
  group_by = c("deploymentID", "locationName"),
  group_time_by = "month"
) %>%
  group_by(month) %>%
  summarise(
    deploymentIDs = list(deploymentID),
    ndep = length(unique(deploymentID)),
    nloc = length(unique(locationName)),
    effort_duration = sum(effort_duration)
  )

```

summarize_observations

Summarize observations information

Description

Summarizes event-based observations by calculating:

- Number of scientific names.
- Number of events.
- Number of observations.
- Sum of individual counts.
- Relative Abundance Index (RAI) based on number of observations.
- Relative Abundance Index (RAI) based on individual counts.

Usage

```

summarize_observations(
  x,
  group_by = c("deploymentID", "latitude", "longitude", "scientificName"),

```

```

    group_time_by = NULL,
    extend = FALSE
  )

  summarise_observations(
    x,
    group_by = c("deploymentID", "latitude", "longitude", "scientificName"),
    group_time_by = NULL,
    extend = FALSE
  )

```

Arguments

<code>x</code>	Camera trap data package object, as returned by <code>camtrapdp::read_camtrapdp()</code> .
<code>group_by</code>	Character vector with names of columns in deployments and observations. At the moment you can choose one or many columns among: <code>c("deploymentID", "latitude", "longitude", "locationID", "locationName", "deploymentStart", "deploymentEnd", "deploymentTags", "scientificName", "lifeStage", "sex", "behavior")</code> . Default: <code>c("deploymentID", "latitude", "longitude", "scientificName")</code> .
<code>group_time_by</code>	Character, one of "day", "week", "month", "year". The effort is calculated at the interval rate defined in <code>group_time_by</code> . Default: NULL, no grouping, i.e. the entire duration of the deployment is taken into account as a whole.
<code>extend</code>	Logical. If TRUE, the summary is extended with all possible groups left out by <code>summarize_observations()</code> . See details section for more information. Default: FALSE.

Details

`summarize_observations()` and `summarise_observations()` are synonyms.

By default (`extend = FALSE`), the function follows the standard behavior of `dplyr::summarise()`, returning only groups that have observations. This means deployments or time periods with zero observations for the specified grouping are excluded from the output.

When `extend = TRUE`, the summary is extended to include all possible combinations of grouping variables, even when no observations exist for a particular group. This is particularly useful for visualisations (`map_summary()`) and analysis as it identifies for example:

- Deployments where a specific species was not observed.
- Time periods when a specific species was not observed.
- Presence/absence patterns across deployments.

For extended summaries, feature values are set to 0 for groups with no observations, except for `n_scientificName` which is set to NA when no species are present as 0 is used when only unidentified individuals are observed.

Value

A grouped tibble data frame with the following columns:

- `group_by` names, e.g. `deploymentID`, `latitude`, `longitude`, and `scientificName`.
- `group_time_by` name if provided, e.g. `month`. It is a datetime column containing the first date of the time interval, e.g. the first day of the month.
- `n_scientificName`: integer vector with the number of scientific names. If `scientificName` is in `group_by`, `n_scientificName` is equal to 1 or 0, if `scientificName` = NA (unidentified animals).
- `n_events`: integer vector with the number of events.
- `n_observations`: integer vector with the number of observations.
- `sum_count`: integer vector with the sum of individual counts.
- `rai_observations`: numeric vector with the Relative Abundance Index (RAI), defined as $100 * (n_observations / effort)$ where `n_observations` is the number of observations and `effort` is the `effort_duration` as returned by `summarize_deployments()` expressed in days.
- `rai_count`: numeric vector with the Relative Abundance Index (RAI), defined as $100 * (sum_count / effort)$ where `sum_count` is the sum of individual counts and `effort` is the `effort_duration` as returned by `summarize_deployments()` expressed in days.

See Also

Other exploration functions: [summarize_deployments\(\)](#)

Examples

```
x <- example_dataset()
# Summarize observations by `deploymentID`, `latitude`, `longitude` and
# `scientificName` (default)
summarize_observations(x)

# Summarize observations by `locationId`, and `locationName` (summary by
# deployment columns only)
summarize_observations(x, group_by = "locationName")

# Summarize observations by `scientificName` and `sex` (summary by
# observation columns only)
summarize_observations(x, group_by = c("scientificName", "sex"))

# Apply temporal grouping by month
summarize_observations(x, group_time_by = "month")

# Extend the summary to include all possible groups
summarize_observations(x, extend = TRUE)
```

`taxa`*Get taxa*

Description

Re-export of `camtrapdp::taxa()`.

Usage

```
taxa(x)
```

Arguments

`x` Camera Trap Data Package object, as returned by `read_camtrapdp()`.

Value

A `tibble::tibble()` data frame with the taxonomic information, containing at least a `scientificName` column.

See Also

Other accessor functions: `contributors()`, `deployments()`, `events()`, `individuals()`, `locations()`, `media()`, `observations()`

Examples

```
x <- example_dataset()
taxa(x)
```

Index

- * **accessor functions**
 - contributors, 8
 - deployments, 9
 - events, 9
 - individuals, 33
 - locations, 34
 - media, 47
 - observations, 49
 - taxa, 57
 - * **datasets**
 - animal_positions, 4
 - calibration_models, 7
 - * **density estimation functions**
 - calculate_individual_radius_angle, 6
 - * **deprecated accessor functions**
 - get_species, 33
 - * **deprecated camtrapR-derived functions**
 - get_cam_op, 15
 - get_detection_history, 18
 - get_record_table, 30
 - * **deprecated density estimation functions**
 - calc_animal_pos, 4
 - * **deprecated exploration functions**
 - get_custom_effort, 17
 - get_effort, 22
 - get_n_individuals, 23
 - get_n_obs, 25
 - get_n_species, 26
 - get_rai, 27
 - get_rai_individuals, 28
 - * **deprecated read functions**
 - read_camtrap_dp, 49
 - * **deprecated visualization functions**
 - map_dep, 35
 - * **exploration functions**
 - summarize_deployments, 52
 - summarize_observations, 54
 - * **filter functions**
 - filter_deployments, 11
 - filter_media, 12
 - filter_observations, 13
 - filter_out_timelapse, 14
 - * **read functions**
 - read_camtrapdp, 51
 - * **read sample data functions**
 - example_dataset, 10
 - mica, 48
 - * **sample data**
 - animal_pos, 3
 - animal_positions, 4
 - calib_models, 7
 - calibration_models, 7
 - mica, 48
 - * **transformation functions**
 - add_coordinates, 3
 - * **visualization functions**
 - map_summary, 41
- add_coordinates, 3
- animal_pos, 3, 4, 7, 48
- animal_positions, 4, 4, 7, 48
- calc_animal_pos, 4
- calculate_individual_radius_angle, 6
- calculate_individual_radius_angle(), 4
- calib_models, 4, 7, 7, 48
- calibration_models, 4, 7, 7, 48
- camtrapdp::contributors(), 8
- camtrapdp::deployments(), 9
- camtrapdp::events(), 9
- camtrapdp::example_dataset(), 10
- camtrapdp::filter_deployments(), 11
- camtrapdp::filter_media(), 12
- camtrapdp::filter_observations(), 13
- camtrapdp::individuals(), 33
- camtrapdp::locations(), 34
- camtrapdp::media(), 47
- camtrapdp::observations(), 49

camtrapdp::read_camtrapdp(), 3, 14, 16, 17, 23–28, 30, 33, 35, 51, 53, 55
 camtrapdp::taxa(), 57
 camtrapR::cameraOperation(), 15, 16
 camtrapR::detectionHistory(), 19
 camtrapR::readCamtrapDP(), 15, 19, 30
 camtrapR::recordTable(), 30
 contributors, 8
 contributors(), 9, 10, 34, 35, 48, 49, 57
 contributors<- (contributors), 8

 deployments, 9
 deployments(), 8, 10, 34, 35, 48, 49, 57
 deployments<- (deployments), 9

 events, 9
 events(), 8, 9, 34, 35, 48, 49, 57
 example_dataset, 10
 example_dataset(), 3, 7, 48

 filter_deployments, 11
 filter_deployments(), 12, 13, 15
 filter_media, 12
 filter_media(), 11, 13, 15
 filter_observations, 13
 filter_observations(), 11, 12, 15, 37, 43
 filter_out_timelapse, 14
 filter_out_timelapse(), 11–13
 frictionless::read_resource(), 51, 52

 get_cam_op, 15
 get_cam_op(), 20, 31
 get_custom_effort, 17
 get_custom_effort(), 23, 24, 26–29
 get_detection_history, 18
 get_detection_history(), 16, 31
 get_effort, 22
 get_effort(), 18, 24, 26–29
 get_n_individuals, 23
 get_n_individuals(), 18, 23, 26–29
 get_n_obs, 25
 get_n_obs(), 18, 23, 24, 27–29
 get_n_species, 26
 get_n_species(), 18, 23, 24, 26, 28, 29
 get_rai, 27
 get_rai(), 18, 23, 24, 26, 27, 29
 get_rai_individuals, 28
 get_rai_individuals(), 18, 23, 24, 26–28
 get_record_table, 30

 get_record_table(), 16, 20
 get_species, 33

 individuals, 33
 individuals(), 8–10, 35, 48, 49, 57

 leaflet::colorNumeric(), 36, 43
 locations, 34
 locations(), 8–10, 34, 48, 49, 57
 lubridate::duration(), 18, 23, 53

 map_dep, 35
 map_summary, 41
 map_summary(), 35
 media, 47
 media(), 8–10, 34, 35, 49, 57
 media<- (media), 47
 mica, 4, 7, 10, 48

 observations, 49
 observations(), 8–10, 34, 35, 48, 57
 observations<- (observations), 49

 read_camtrap_dp, 49
 read_camtrapdp, 51
 read_camtrapdp(), 8–13, 33, 34, 48, 49, 57

 summarise_deployments
 (summarize_deployments), 52
 summarise_observations
 (summarize_observations), 54
 summarize_deployments, 52
 summarize_deployments(), 17, 22, 35, 36, 42, 56
 summarize_observations, 54
 summarize_observations(), 23, 25–27, 29, 35, 42, 53

 taxa, 57
 taxa(), 8–10, 33–35, 48–50, 52
 tibble::tibble(), 8–10, 34, 48, 49, 57

 unmarked::unmarkedMultFrame(), 19