Package 'camtrapdp'

January 14, 2026

Title Read and Manipulate Camera Trap Data Packages

Version 0.5.0

Date 2026-01-13

Description Read and manipulate Camera Trap Data Packages ('Camtrap DP'). 'Camtrap DP' (<https://camtrap-dp.tdwg.org>) is a data exchange format for camera trap data. With 'camtrapdp' you can read, filter and transform data (including to Darwin Core) before further analysis in e.g. 'camtraptor' or 'camtrapR'.

License MIT + file LICENSE

URL https://github.com/inbo/camtrapdp, https://inbo.github.io/camtrapdp/

BugReports https://github.com/inbo/camtrapdp/issues

Depends R (>= 3.6.0)

Imports cli, dplyr, EML, frictionless (>= 1.2.1), memoise, purrr, readr, rlang, stringr, uuid

Suggests jsonlite, lubridate, testthat (>= 3.0.0), tibble, xml2

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Peter Desmet [aut, cre] (ORCID: <https://orcid.org/0000-0002-8442-8025>, affiliation: Research Institute for Nature and Forest (INBO)), Sanne Govaert [aut] (ORCID: <https://orcid.org/0000-0002-8939-1305>, affiliation: Research Institute for Nature and Forest (INBO)), Pieter Huybrechts [aut] (ORCID: <https://orcid.org/0000-0002-6658-6062>, affiliation: Research Institute for Nature and Forest (INBO)), Damiano Oldoni [aut] (ORCID: <https://orcid.org/0000-0003-3445-7562>, affiliation: Research Institute for Nature and Forest (INBO)), Research Institute for Nature and Forest (INBO) [cph] (ROR:

<https://ror.org/00j54wy13>),
Research Foundation - Flanders [fnd] (https://lifewatch.be)

Maintainer Peter Desmet <peter.desmet@inbo.be>

Repository CRAN

Date/Publication 2026-01-14 14:10:03 UTC

Contents

`check_camtrapdp` *Check a Camera Trap Data Package object*

Description

Checks if an object is a Camera Trap Data Package object with the required properties.

Usage

```
check_camtrapdp(x)
```

Arguments

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

Value

x invisibly or an error.

Examples

```
x <- example_dataset()
check_camtrapdp(x) # Invisible return of x if valid
```

---

contributors *Get or set contributors*

Description

`contributors()` gets contributors from the `x$contributors` property in a Camera Trap Data Package object and returns it as a tibble data frame.
`contributors()<-` is the assignment equivalent.

Usage

```
contributors(x)

contributors(x) <- value
```

Arguments

x	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
value	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` *Get or set deployments*

Description

`deployments()` gets the deployments from a Camera Trap Data Package object.
`deployments()<-` is the assignment equivalent.

- It should only be used within other functions, where the expected data structure can be guaranteed.
- Metadata (`x$spatial` and `x$temporal`) are updated to match the assigned deployments.

Usage

```
deployments(x)

deployments(x) <- value
```

Arguments

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

Description

Gets the (unique) events from the observations of a Camera Trap Data Package object. Only observations with `observationLevel == "event"` are considered.

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

Reads the Camtrap DP example dataset. This dataset is maintained and versioned with the Camtrap DP standard.

Usage

```
example_dataset()
```

Value

Camera Trap Data Package object.

Examples

```
example_dataset()
```

filter_deployments *Filter deployments*

Description

Subsets deployments in a Camera Trap Data Package object, retaining all rows that satisfy the conditions.

- Media are filtered on associated `deploymentID`.
- Observations are filtered on associated `deploymentID`.
- Metadata (`x$spatial`, `x$temporal` and `x$taxonomic`) are updated to match the filtered deployments.

Usage

```
filter_deployments(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_media()`, `filter_observations()`

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

`filter_media` *Filter media*

Description

Subsets media in a Camera Trap Data Package object, retaining all rows that satisfy the conditions.

- Deployments are not filtered.
- Observations are filtered on associated `mediaID` (for media-based observations) and `eventID` (for event-based observations).
- Metadata (`x$taxonomic`) are updated to match the filtered observations.

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()`

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

`filter_observations` *Filter observations*

Description

Subsets observations in a Camera Trap Data Package object, retaining all rows that satisfy the conditions.

- Deployments are not filtered.

- Media are filtered on associated `mediaID` (for media-based observations) and `eventID` (for event-based observations). Filter on `observationLevel == "media"` to only retain directly linked media.
- Metadata (`x$taxonomic`) are updated to match the filtered observations.

Usage

```
filter_observations(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_media()`

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

`individuals` *Get individuals*

Description

Gets the (unique) individuals from the observations of a Camera Trap Data Package object.

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

Gets the (unique) locations from the deployments of a Camera Trap Data Package object.

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)
```

`media` *Get or set media*

Description

`media()` gets the media from a Camera Trap Data Package object.
`media()<-` is the assignment equivalent.

- It should only be used within other functions, where the expected data structure can be guaranteed.

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)
```

`merge_camtrapdp` *Merge two Camera Trap Data Packages*

Description

Merges two Camera Trap Data Package objects into one. Repeat to merge multiple datasets.

Usage

```
merge_camtrapdp(x, y)
```

Arguments

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

Value

A single Camera Trap Data Package object that is the combination of `x` and `y`.

Transformation details

Both `x` and `y` must have a unique dataset name `x$name` and `y$name`. This name is used to prefix identifiers in the data that occur in both datasets. For example:

- `x` contains `deploymentIDs` `c("a", "b")`.
- `y` contains `deploymentIDs` `c("b", "c")`.
- Then merged `xy` will contain `deploymentIDs` `c("a", "x_b", "y_b", "c")`.

Data are merged as follows:

- Deployments are combined, with `deploymentID` kept unique.
- Media are combined, with `mediaID`, `deploymentID` and `eventID` kept unique.
- Observations are combined, with `observationID`, `deploymentID`, `mediaID` and `eventID` kept unique.
- Additional resources are retained, with the resource name kept unique.

Metadata properties are merged as follows:

- **name**: Removed.
- **id**: Removed.
- **created**: Set to current timestamp.
- **title**: Removed.
- **contributors**: Combined, with duplicates removed.
- **description**: Combined as two paragraphs.
- **version**: Set to `1.0`.

- **keywords**: Combined, with duplicates removed.
- **image**: Removed.
- **homepage**: Removed.
- **sources**: Combined, with duplicates removed.
- **licenses**: Combined, with duplicates removed.
- **bibliographicCitation**: Removed.
- **project$id**: Removed.
- **project$title**: Combined.
- **project$acronym**: Removed.
- **project$description**: Combined as two paragraphs.
- **project$path**: Removed.
- **project$samplingDesign**: Sampling design of `x`.
- **project$captureMethod**: Combined, with duplicates removed.
- **project$individuals**: `TRUE` if one of the datasets has `TRUE`.
- **project$observationLevel**: Combined, with duplicates removed.
- **coordinatePrecision**: Set to the least precise `coordinatePrecision`.
- **spatial**: Reset based on the new deployments.
- **temporal**: Reset based on the new deployments.
- **taxonomic**: Combined, with duplicates removed.
- **relatedIdentifiers**: Combined, with duplicates removed.
- **references**: Combined, with duplicates removed.
- Custom properties of `x` are also retained.

See Also

Other transformation functions: `round_coordinates()`, `shift_time()`, `update_taxon()`, `write_dwc()`, `write_eml()`

Examples

```
x <- example_dataset() %>%
  filter_deployments(deploymentID %in% c("00a2c20d", "29b7d356"))
y <- example_dataset() %>%
  filter_deployments(deploymentID %in% c("577b543a", "62c200a9"))
x$name <- "x"
y$name <- "y"
merge_camtrapdp(x, y)
```

`observations` *Get or set observations*

Description

`observations()` gets the observations from a Camera Trap Data Package object.
`observations()<-` is the assignment equivalent.

- It should only be used within other functions, where the expected data structure can be guaranteed.
- Metadata (`x$taxonomic`) are updated to match the assigned observations.

Usage

```
observations(x)

observations(x) <- value
```

Arguments

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

print.camtrapdp *Print a Camera Trap Data Package*

Description

Prints a human-readable summary of a Camera Trap Data Package, as an extension of `frictionless::print.datapackage(`

Usage

```
## S3 method for class 'camtrapdp'
print(x, ...)
```

Arguments

x	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
...	Further arguments, they are ignored by this function.

Value

`print()` with a summary of the Camera Trap Data Package object.

Examples

```
x <- example_dataset()

# Print a summary
print(x)

# Print a summary after filtering
filter_deployments(x, deploymentID == "62c200a9")
```

read_camtrapdp *Read a Camera Trap Data Package*

Description

Reads a Camera Trap Data Package (Camtrap DP) dataset into memory.

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** `eventID`**s 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
```

`round_coordinates` *Round coordinates to generalize camera trap locations*

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(x, digits)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`digits`	Number of decimal places to round coordinates to (`1`, `2` or `3`).

Value

`x` with chosen `coordinatePrecision` in metadata and rounded coordinates and calculated `coordinateUncertainty` in deployments.

Details

Rounding coordinates is a recommended method to generalize sensitive biodiversity information (see Section 4.2 in Chapman 2020). Use this function to do so for your data. Determine the category of sensitivity (see Section 2.2 in Chapman 2020) and choose the associated number of digits :

category	sensitivity	digits
category 1	extreme	(do not publish)
category 2	high	1
category 3	medium	2
category 4	low	3
not sensitive	not sensitive	all (do not use this function)

The function will:

1. Set the `coordinatePrecision` in the metadata (original values will be overwritten):

digits	coordinatePrecision
1	0.1
2	0.01
3	0.001

2. Round all coordinates in the deployments to the selected number of digits.
3. Update the `coordinateUncertainy` (in meters) in the deployments. This uncertainty is based on the number of digits and the latitude, following Table 3 in Chapman & Wieczorek 2020:

digits	0° latitude	30° latitude	60° latitude	85° latitude
1	15691 m	14697 m	12461 m	11211 m
2	1570 m	1470 m	1246 m	1121 m
3	157 m	147 m	125 m	112 m

If a `coordinatePrecision` is already present, the function will subtract the `coordinateUncertainty` associated with it before setting a new uncertainty (e.g. `0.001` to `0.01` = `original value - 157 + 1570 m`). If `original value` is `NA`, the function will assume the coordinates were obtained by GPS and set `original value = 30`.

See Also

Other transformation functions: `merge_camtrapdp()`, `shift_time()`, `update_taxon()`, `write_dwc()`, `write_eml()`

Examples

```
x <- example_dataset()

# Original precision
x$coordinatePrecision

# Original coordinates and uncertainty
deployments(x)[c("latitude", "longitude", "coordinateUncertainty")]

# Round coordinates to 1 digit
x_rounded <- round_coordinates(x, 1)

# Updated coordinatePrecision
x_rounded$coordinatePrecision

# Updated coordinates and uncertainty (original 187 - 147 + 14697 = 14737)
deployments(x_rounded)[c("latitude", "longitude", "coordinateUncertainty")]
```

shift_time *Shift date-times*

Description

Shifts date-times for selected deployments (and associated media and observations) by a specified duration. This function can be used to correct date-time issues such as incorrectly set time zones.

- Deployments: `deploymentStart` and `deploymentEnd` are updated and `timestampIssues` is set to `FALSE`.
- Media: `timestamp` is updated.
- Observations: `eventStart` and `eventEnd` are updated.
- Metadata (`x$temporal`) are updated to match the new temporal scope.

Usage

```
shift_time(x, deployment_id, duration)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`deployment_id`	One or more deploymentIDs.
`duration`	Difference between the current and new date-times. Provide as a `lubridate::duration()` or difftime.

Value

`x` with shifted date-times.

See Also

Other transformation functions: `merge_camtrapdp()`, `round_coordinates()`, `update_taxon()`, `write_dwc()`, `write_eml()`

Examples

```
# Set desired duration between current and new date-times (e.g. 4 hours earlier)
library(lubridate, warn.conflicts = FALSE)
duration(-4, units = "hours")

# Or calculate one based on two date-times
current <- ymd_hms("2024-04-01T04:00:00", tz = "UTC")
new <- ymd_hms("2024-04-01T00:00:00", tz = "UTC")
duration <- as.duration(interval(current, new))

# Shift date-times for 2 deployments
x <- example_dataset()
```

```
x_shifted <- shift_time(x, c("00a2c20d", "29b7d356"), duration)

# Inspect results
deployments(x)[, c("deploymentID", "deploymentStart", "deploymentEnd")]
deployments(x_shifted)[, c("deploymentID", "deploymentStart", "deploymentEnd")]
```

---

taxa *Get taxa*

Description

Gets the (unique) scientific names and associated taxonomic information from the observations of a Camera Trap Data Package object. Duplicate taxa (i.e. with the same `scientificName`) are removed, retaining the taxon with (first) a `taxonID` and (second) the most taxonomic information.

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)
```

update_taxon *Update a taxon*

Description

Updates taxonomic information in data and metadata for a provided taxon. This allows to:

1. Update a taxon: provide the same name in `to` and `from$scientificName`.
2. Replace a taxon: provide a new name in `from$scientificName`.
3. Lump a taxon: provide a name in `from$scientificName` that is already present in the dataset. In all cases, existing information will be overwritten with the provided information.

Usage

```
update_taxon(x, from, to)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`from`	`scientificName` of the taxon to update.
`to`	Named list with taxon information, e.g. `list(scientificName = "Ardea", taxonRank = "genus", vernacularname.eng = "great herons")`.

Value

`x` with updated taxon information.

See Also

Other transformation functions: `merge_camtrapdp()`, `round_coordinates()`, `shift_time()`, `write_dwc()`, `write_eml()`

Examples

```
x <- example_dataset()

# Update taxonomic information for "Anas platyrhynchos"
updated_x <- update_taxon(
  x,
  from = "Anas platyrhynchos",
  to = list (
    scientificName = "Anas platyrhynchos",
    taxonID = "https://www.checklistbank.org/dataset/COL2023/taxon/DGP6",
    taxonRank = "species",
    vernacularNames.fra = "canard colvert"
  )
)
```

```
# Lump "Ardea cinerea" into already present "Ardea", using the provided info
updated_x <- update_taxon(
  x,
  from = "Ardea cinerea",
  to = list(scientificName = "Ardea", vernacularname.fra = "grands hérons")
)
```

`version` *Get Camtrap DP version*

Description

Extracts the version number used by a Camera Trap Data Package object. This version number indicates what version of the Camtrap DP standard was used.

Usage

```
version(x)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`. Also works on a Frictionless Data Package, as returned by `frictionless::read_package()`.

Details

The version number is derived as follows:

1. The `version` attribute, if defined.
2. A version number contained in `x$profile`, which is expected to contain the URL to the used Camtrap DP standard.
3. `x$profile` in its entirety (can be `NULL`).

Value

Camtrap DP version number (e.g. `1.0`).

Examples

```
x <- example_dataset()
version(x)
```

`write_camtrapdp` *Write a Camera Trap Data Package to disk*

Description

Writes a Camera Trap Data Package and its related Data Resources to disk as a `datapackage.json` and CSV files.

Usage

```
write_camtrapdp(x, directory, ...)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`directory`	Path to local directory to write files to.
`...`	Further arguments, passed to `frictionless::write_package()` (e.g. `compress = TRUE`).

Value

`datapackage.json` and CSV files written to disk.

Examples

```
x <- example_dataset()

# Filter (and therefore change) the dataset
x <- filter_deployments(x, deploymentID == "00a2c20d")

# Write the Camera Trap Data Package to disk
write_camtrapdp(x, directory = "my_directory")

# Check files
list.files("my_directory")

# Clean up (don't do this if you want to keep your files)
unlink("my_directory", recursive = TRUE)
```

write_dwc *Transform a Camera Trap Data Package to a Darwin Core Archive*

Description

Transforms a Camera Trap Data Package object to a Darwin Core Archive.

Usage

```
write_dwc(x, directory)
```

Arguments

x	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
directory	Path to local directory to write files to.

Value

CSV and `meta.xml` files written to disk. And invisibly, a list of data frames with the transformed data.

Transformation details

This function **follows recommendations** in Reyserhove et al. (2023) doi:10.35035/doc0qzp2x37 and transform data to:

- An Occurrence core.
- An Audubon/Audiovisual Media Description extension.
- A `meta.xml` file.

Key features of the Darwin Core transformation:

- The Occurrence core contains one row per observation (`dwc:occurrenceID` = `observationID`).
- Only observations with `observationType` = `"animal"` and are included, thus excluding observations that are (of) humans, vehicles, blanks, unknowns and unclassified.
- Either observations with `observationLevel` = `"event"` or `"media"` are used, never both to avoid duplicates. The level be defined with `x$gbifIngestion$observationLevel`, with `"event"` as default.
- Observations classified by humans with 100% certainty get a `dwc:identificationVerificationStatus` = `"verified using recorded media"`.
- Deployment information is included in the Occurrence core, such as location, habitat, `dwc:samplingProtocol`, deployment duration in `dwc:samplingEffort` and `dwc:parentEventID` = `deploymentID` as grouping identifier.
- Event information is included in the Occurrence core, as event duration in `dwc:eventDate` and `dwc:eventID` = `eventID` as grouping identifier.

- Media files are included in the Audubon/Audiovisual Media Description extension, with a foreign key to the observation. A media file that is used for more than one observation is repeated.
- Metadata are used to set the following record-level terms:
  - `dwc:datasetID`: `x$id`.
  - `dwc:datasetName`: `x$title`.
  - `dwc:collectionCode`: first source in `x$sources`.
  - `dcterms:license`: license `name` (e.g. `CC0-1.0`) in `x$licenses` with scope `data`. The license `name` with scope `media` is used as `dcterms:rights` in the Audubon Media Description extension.
  - `dcterms:rightsHolder`: first contributor in `x$contributors` with role `rightsHolder`.
  - `dwc:dataGeneralizations`: set if `x$coordinatePrecision` is defined.

See Also

Other transformation functions: `merge_camtrapdp()`, `round_coordinates()`, `shift_time()`, `update_taxon()`, `write_eml()`

Examples

```
x <- example_dataset()
write_dwc(x, directory = "my_directory")

# Clean up (don't do this if you want to keep your files)
unlink("my_directory", recursive = TRUE)
```

`write_eml` *Transform a Camera Trap Data Package to EML*

Description

Transforms the metadata of a Camera Trap Data Package object to an Ecological Metadata Language (EML) file.

Usage

```
write_eml(x, directory, derived_paragraph = TRUE)
```

Arguments

`x`	Camera Trap Data Package object, as returned by `read_camtrapdp()`.
`directory`	Path to local directory to write files to.
`derived_paragraph`	If `TRUE`, a paragraph will be added to the abstract, indicating that data have been transformed using `write_dwc()`.

Value

`eml.xml` file written to disk. And invisibly, an EML::eml object.

Transformation details

Metadata are derived from what is provided in `x`. The following properties are set:

- **title**: Title as provided in `x$title`.
- **type**: Set to `Occurrence` in keywords.
- **subtype**: Set to `Observation` in keywords.
- **update frequency**: Set to `unknown`.
- **description**: Description as provided in `x$description`. If `derived_paragraph = TRUE` a generated paragraph is added, e.g.:

  Data have been standardized to Darwin Core using the camtrapdp R package and only include observations (and associated media) of animals. Excluded are records that document blank or unclassified media, vehicles and observations of humans.
- **license**: License with scope `data` as provided in `x$licenses`.
- **creators**: Contributors as provided in `x$contributors`, excluding those with roles `rightsHolder` and `publisher`.
- **contact**: Contributors with role `contact`. If none exist, first creator.
- **metadata provider**: Same as `contact`.
- **keywords**: Keywords as provided in `x$keywords`.
- **geographic coverage**: Bounding box as provided in `x$spatial`.
- **taxonomic coverage**: Taxa as provided in `x$taxonomic`.
- **temporal coverage**: Date range as provided in `x$temporal`.
- **project data**: Title, acronym as identifier, description, and sampling design as provided in `x$project`.
- **alternative identifier**: Identifier as provided in `x$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 `x$project$path`.

The following properties are not set:

- **publishing organization**
- **associated parties**
- **sampling methods**
- **citations**
- **collection data**: not applicable.

See Also

Other transformation functions: `merge_camtrapdp()`, `round_coordinates()`, `shift_time()`, `update_taxon()`, `write_dwc()`

Examples

```
x <- example_dataset()
(write_eml(x, directory = "my_directory"))

# Clean up (don't do this if you want to keep your files)
unlink("my_directory", recursive = TRUE)
```

Index