Package: R2camtrapdp (via r-universe)

July 1, 2026

Type Package

Title Convert Camera Trap Dataset to 'Camtrap DP'

Version 2.0.0

Description Builds Camera Trap Data Packages ('Camtrap DP') from arbitrary spreadsheets in a schema-driven way: table structure, types, constraints and relations are read from the 'Frictionless' table schemas of the requested 'Camtrap DP' version, so any version and custom columns are handled automatically. Provides validation against the schemas and an optional bridge to the 'frictionless' 'Python' framework. The 'Camtrap DP' standard is described in Bubnicki et al. (2023) <doi:10.1002/rse2.374>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Imports R6, jsonlite, tibble, magrittr, lubridate, dplyr, tidyr, purrr, readr, httr, taxadb, stats, utils

Suggests camtrapdp, knitr, rmarkdown, testthat (>= 3.0.0), jsonvalidate, spelling

VignetteBuilder knitr

Depends R (>= 3.5.0)

Config/roxygen2/version 8.0.0

SystemRequirements Python (>= 3.8) with the 'frictionless' package (optional; only for validate_frictionless())
Config/testthat/edition: 3

URL https://github.com/kfukasawa37/R2camtrapdp

BugReports https://github.com/kfukasawa37/R2camtrapdp/issues

Language en-US

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev python3 libx11-dev xz-utils

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

Date/Publication 2026-06-23 02:21:13 UTC

RemoteUrl https://github.com/kfukasawa37/R2camtrapdp

RemoteRef HEAD

RemoteSha a37dfd8fcb790bcd2b68a7f34e00a2d348f4044e

Contents

Adep . . . 3
Aobs . . . 4
create_deployments . . . 5
create_media . . . 7
create_observations . . . 8
ctdp-build-table . . . 11
ctdp-conformance . . . 11
ctdp-frictionless . . . 11
ctdp-mapping . . . 11
ctdp-references . . . 12
ctdp-validation-report . . . 12
ctdp_apply_mapping . . . 12
ctdp_bind_issues . . . 13
ctdp_build_table . . . 13
ctdp_check_schema . . . 15
ctdp_is_valid . . . 15
ctdp_issues . . . 16
ctdp_merge_datetime . . . 17
ctdp_no_issues . . . 18
ctdp_parse_frictionless . . . 18
ctdp_schema_references . . . 19
ctdp_semantic_only_fields . . . 20
ctdp_summarize_validation . . . 20
ctdp_validate_frictionless . . . 21
datapackageAdata . . . 22
datapackageIdata . . . 22
datapackageVdata . . . 23
Idep . . . 23
Iobs . . . 24
MetadataProfile . . . 25
R6_CamtrapDP . . . 27
TableSchema . . . 39
Vdep . . . 43
Vobs . . . 44

Index **46**

Adep *Example acoustic deployment field-notebook*

Description

Example deployment notebook for an acoustic (audio) survey, used in the acoustic vignette. One row per device deployment. Coordinates are random points within the inland Izu Peninsula (Shizuoka, Japan).

Usage

```
Adep
```

Format

A data frame with 2 rows and 14 variables:

deploymentID Unique identifier of the deployment.

longitude Longitude in decimal degrees (WGS84).

latitude Latitude in decimal degrees (WGS84).

locationID Identifier of the deployment location.

startDate Deployment start date.

startTime Deployment start time.

endDate Deployment end date.

endTime Deployment end time.

deviceID Identifier of the recording device.

deviceModel Manufacturer and model of the recording device.

samplingFrequency Sampling frequency of the recordings (Hz).

bitDepth Bit depth of the recordings.

channels Number of audio channels.

setupBy Name or identifier of the person/organization that deployed the device.

See Also

Aobs

`Aobs` *Example acoustic observation field-notebook*

Description

Example observation notebook for an acoustic (audio) survey, used in the acoustic vignette. One row per observation; the `filename` column is the audio file from which the `media` table is derived.

Usage

```
Aobs
```

Format

A data frame with 6 rows and 19 variables:

institutionCode Institution code.

collectionCode Collection code.

obsID Observation identifier (within an event).

eventID Identifier of the event the observation belongs to.

deploymentID Identifier of the deployment.

locationID Identifier of the deployment location.

date Date the recording was made.

time Time the recording started.

filename Name of the audio file (used to build `media`).

duration Duration of the recording file (seconds).

object Recorded object category (`animal`, `none`, ...).

class Taxonomic class of the observed organism.

genus Genus of the observed organism.

species Species epithet of the observed organism.

individualCount Number of observed individuals; `NA` here (not counted from audio).

frequencyLow Lower bound of the call frequency (Hz); approximate values from the bioacoustics literature for each species.

frequencyHigh Upper bound of the call frequency (Hz); approximate values from the bioacoustics literature for each species.

eventStart Date and time at which the event started.

eventEnd Date and time at which the event ended.

See Also

Adep

`create_deployments` *Create deployments*

Description

`create_deployments` creates a table of deployments.

Usage

```
create_deployments(
  deploymentID,
  latitude,
  longitude,
  deploymentStart = NULL,
  deploymentStart_date = NULL,
  deploymentStart_time = NULL,
  deploymentEnd = NULL,
  deploymentEnd_date = NULL,
  deploymentEnd_time = NULL,
  locationID = NULL,
  locationName = NULL,
  coordinateUncertainty = NULL,
  setupBy = NULL,
  cameraID = NULL,
  cameraModel = NULL,
  cameraDelay = NULL,
  cameraHeight = NULL,
  cameraDepth = NULL,
  cameraTilt = NULL,
  cameraHeading = NULL,
  detectionDistance = NULL,
  timestampIssues = NULL,
  baitUse = NULL,
  featureType = NULL,
  habitat = NULL,
  deploymentGroups = NULL,
  deploymentTags = NULL,
  deploymentComments = NULL,
  tz = "Asia/Tokyo"
)
```

Arguments

`deploymentID`	Unique identifier of the deployment
`latitude`	Latitude of the deployment location in decimal degrees, using the WGS84 datum
`longitude`	Longitude of the deployment location in decimal degrees, using the WGS84 datum

`deploymentStart`
: Date and time at which the deployment was started

`deploymentStart_date`
: Date at which the deployment was started

`deploymentStart_time`
: Time at which the deployment was started

`deploymentEnd`
: Date and time at which the deployment was ended

`deploymentEnd_date`
: Date at which the deployment was ended

`deploymentEnd_time`
: Time at which the deployment was ended

`locationID`
: Identifier of the deployment location

`locationName`
: Name given to the deployment location

`coordinateUncertainty`
: Horizontal distance from the given latitude and longitude describing the smallest circle containing the deployment location

`setupBy`
: Name or identifier of the person or organization that deployed the camera

`cameraID`
: Identifier of the camera used for the deployment

`cameraModel`
: Manufacturer and model of the camera

`cameraDelay`
: Predefined duration after detection when further activity is ignored

`cameraHeight`
: Height at which the camera was deployed

`cameraDepth`
: Depth at which the camera was deployed

`cameraTilt`
: Angle at which the camera was deployed in the vertical plane

`cameraHeading`
: Angle at which the camera was deployed in the horizontal plane

`detectionDistance`
: Maximum distance at which the camera can reliably detect activity

`timestampIssues`
: true if timestamps in the media resource for the deployment are known to have unsolvable issues

`baitUse`
: true if bait was used for the deployment

`featureType`
: Type of the feature associated with the deployment

`habitat`
: Short characterization of the habitat at the deployment location

`deploymentGroups`
: Deployment groups associated with the deployment

`deploymentTags`
: Tags associated with the deployment

`deploymentComments`
: Comments or notes about the deployment

`tz`
: Deployment time zone

Value

A tibble of deployments.

Examples

```
create_deployments(
  deploymentID = c("A01", "A02"),
  latitude = c(35.1, 36.2), longitude = c(139.5, 140.1),
  deploymentStart_date = c("2023-04-01", "2023-04-02"),
  deploymentStart_time = c("09:00:00", "10:30:00"),
  deploymentEnd_date = c("2023-05-01", "2023-05-02"),
  deploymentEnd_time = c("09:00:00", "10:30:00"))
```

---

create_media *Create media*

Description

`create_media` creates a table of media.

Usage

```
create_media(
  mediaID,
  deploymentID,
  timestamp = NULL,
  timestamp_date = NULL,
  timestamp_time = NULL,
  filePath,
  filePublic,
  fileMediatype,
  fileName = NULL,
  captureMethod = NULL,
  exifData = NULL,
  favorite = NULL,
  mediaComments = NULL,
  tz = "Asia/Tokyo",
  omitduplicate = TRUE
)
```

Arguments

`mediaID`	Unique identifier of the media file
`deploymentID`	Identifier of the deployment the media file belongs to
`timestamp`	Date and time at which the media file was recorded
`timestamp_date`	Date at which the media file was recorded
`timestamp_time`	Time at which the media file was recorded
`filePath`	URL or relative path to the media file, respectively for externally hosted files or files that are part of the package

filePublic	false if the media file is not publicly accessible
fileMediatype	Mediatype of the media file. Expressed as an IANA Media Type
fileName	Name of the media file
captureMethod	Method used to capture the media file
exifData	EXIF data of the media file
favorite	true if the media file is deemed of interest
mediaComments	Comments or notes about the media file
tz	Time zone of the media file was recorded
omitduplicate	true if duplicate exclusion

Value

A tibble of media.

Examples

```
create_media(
  mediaID = "m1", deploymentID = "A01",
  timestamp_date = "2023-04-01", timestamp_time = "09:05:00",
  filePath = "img/m1.jpg", filePublic = TRUE, fileMediatype = "image/jpeg")
```

---

create_observations *Create observations*

Description

create_media creates a table of observations.

Usage

```
create_observations(
  observationID,
  deploymentID,
  mediaID = NULL,
  eventID = NULL,
  eventStart = NULL,
  eventStart_date = NULL,
  eventStart_time = NULL,
  eventEnd = NULL,
  eventEnd_date = NULL,
  eventEnd_time = NULL,
  observationLevel,
  observationType,
  cameraSetupType = NULL,
  scientificName = NULL,
```

```
  count = NULL,
  lifeStage = NULL,
  sex = NULL,
  behavior = NULL,
  individualID = NULL,
  individualPositionRadius = NULL,
  individualPositionAngle = NULL,
  individualSpeed = NULL,
  bboxX = NULL,
  bboxY = NULL,
  bboxWidth = NULL,
  bboxHeight = NULL,
  classificationMethod = NULL,
  classifiedBy = NULL,
  classificationTimestamp = NULL,
  classificationProbability = NULL,
  observationTags = NULL,
  observationComments = NULL,
  tz = "Asia/Tokyo",
  omitduplicate = TRUE
)
```

Arguments

observationID	Unique identifier of the observation
deploymentID	Identifier of the deployment the observation belongs to
mediaID	Identifier of the media file that was classified
eventID	Identifier of the event the observation belongs to
eventStart	Date and time at which the event started
eventStart_date	Date at which the event started
eventStart_time	Time at which the event started
eventEnd	Date and time at which the event ended
eventEnd_date	Date at which the event ended
eventEnd_time	Time at which the event ended
observationLevel	Level at which the observation was classified
observationType	Type of the observation
cameraSetupType	Type of the camera setup action associated with the observation
scientificName	Scientific name of the observed individual
count	Number of observed individuals
lifeStage	Age class or life stage of the observed individual

sex	Sex of the observed individual
behavior	Dominant behavior of the observed individual
individualID	Identifier of the observed individual
individualPositionRadius	Distance from the camera to the observed individual identified by individualID
individualPositionAngle	Angular distance from the camera view centerline to the observed individual identified by individualID
individualSpeed	Average movement speed of the observed individual identified by individualID
bboxX	Horizontal position of the top-left corner of a bounding box
bboxY	Vertical position of the top-left corner of a bounding box
bboxWidth	Width of a bounding box
bboxHeight	Height of the bounding box
classificationMethod	Method used to classify the observation
classifiedBy	Name or identifier of the person or AI algorithm that classified the observation
classificationTimestamp	Date and time of the classification
classificationProbability	Degree of certainty of the classification
observationTags	Tags associated with the observation
observationComments	Comments or notes about the observation
tz	Time zone of observation
omitduplicate	true if duplicate exclusion

Value

A tibble of observations.

Examples

```
create_observations(
  observationID = "o1", deploymentID = "A01", mediaID = "m1",
  eventStart = "2023-04-01 09:05:00", eventEnd = "2023-04-01 09:05:00",
  observationLevel = "media", observationType = "animal",
  scientificName = "Vulpes vulpes", count = 1L)
```

`ctdp-build-table` *Build a schema-conformant table from arbitrary input*

Description

`ctdp_build_table()` is the generic, schema-driven path from an arbitrary input spreadsheet to a Camtrap DP table. It (1) applies an optional column mapping, (2) optionally merges separate date/time columns into datetime columns, (3) coerces columns to the schema types, and (4) validates the result against the schema constraints. It works for any Camtrap DP version and for custom columns, because everything is read from the supplied TableSchema.

`ctdp-conformance` *Frictionless conformance pre-checks (R-side)*

Description

The R-side validation in this package normally *trusts* that the supplied table schema is a well-formed Frictionless Table Schema. These helpers add an R-side pre-check of that assumption – catching, before the Python Frictionless step, the kinds of structural problems Frictionless itself rejects (unsupported field `type`, a `constraint` that is not valid for a field's type, primary/foreign keys that reference undefined fields, etc.). The authoritative check remains R6_CamtrapDP's `validate_frictionless()`.

`ctdp-frictionless` *Python Frictionless validation (shared internals + validate-only API)*

Description

Helpers to run the Python `frictionless` validator on a written data package and parse the report. `ctdp_validate_frictionless()` validates an **existing** data package directory **without writing or overwriting** anything (use this when you only want to validate a package that was created elsewhere). The R6_CamtrapDP method `validate_frictionless()` reuses the same internals but writes the package from the object first (its `write` argument).

`ctdp-mapping` *Column mapping helpers*

Description

Map an arbitrary input spreadsheet onto Camtrap DP field names, and combine separate date / time columns into a single datetime column.

ctdp-references *Discover external references inside a schema*

Description

Camtrap DP schemas specify some information not through machine-enforceable Frictionless constraints, but through **URLs**: semantic mappings (`skos:exactMatch` / `skos:broadMatch` / `skos:narrowMatch` to Darwin Core, Audubon Core, Dublin Core, ... terms), reference URLs embedded in field `descriptions` (e.g. the IANA media-type registry for `fileMediatype`, or method DOIs for `individualSpeed`), and the resource `schema` / package `profile` URLs themselves.

These functions surface every such URL so that, when you adopt a new version or a new schema flavor, you do not overlook a specification that is expressed only by reference.

ctdp-validation-report *Validation issue representation and reporting*

Description

A uniform structure for validation issues, whether they are produced by the R-side schema/relation checks or parsed from a Python Frictionless validation report. An "issue table" is a tibble::tibble with the columns described in `ctdp_issues()`.

ctdp_apply_mapping *Apply a column mapping to a data frame*

Description

Renames the columns of `df` from the source (spreadsheet) names to Camtrap DP field names. Columns of `df` that are not mentioned in `mapping` are kept unchanged, so columns that already use Camtrap DP field names pass through.

Usage

```
ctdp_apply_mapping(df, mapping, drop_unmapped = FALSE)
```

Arguments

`df`	A data.frame / tibble of input data.
`mapping`	The mapping from source column names to target field names. Accepted forms: • a named character vector where **names are source columns** and **values are target fields**, e.g. `c(lat = "latitude", lon = "longitude")`; • a data.frame / tibble with columns `source` and `target`.
`drop_unmapped`	If `TRUE`, keep only the mapped target columns; if `FALSE` (default) also keep unmapped columns as-is.

Value

A tibble with renamed columns.

Examples

```
raw <- data.frame(station = c("A01", "A02"), lat = c(35.1, 36.2))
ctdp_apply_mapping(raw, c(station = "deploymentID", lat = "latitude"))
```

---

ctdp_bind_issues *Row-bind several issue tables*

Description

Row-bind several issue tables

Usage

```
ctdp_bind_issues(...)
```

Arguments

...	Issue tables (tibbles), or NULLs which are dropped.

Value

A single combined issue tibble.

Examples

```
a <- ctdp_issues(source = "media", constraint = "required",
                 severity = "error", message = "missing mediaID")
ctdp_bind_issues(a, ctdp_no_issues())
```

---

ctdp_build_table *Build and validate a table against a Table Schema*

Description

Build and validate a table against a Table Schema

Usage

```
ctdp_build_table(
  schema,
  data,
  mapping = NULL,
  datetime_merges = NULL,
  tz = "Asia/Tokyo",
  source = NULL,
  coerce = TRUE,
  stop_on_error = FALSE
)
```

Arguments

schema	A TableSchema object.
data	A data.frame / tibble of input data.
mapping	Optional column mapping; see ctdp_apply_mapping().
datetime_merges	Optional list of date/time merge specs. Each element is a list with date_col, time_col, target (and optionally format), applied with ctdp_merge_datetime() **after** mapping.
tz	Time zone used for temporal coercion / merging.
source	Issue source label; defaults to "<resource>.csv".
coerce	If TRUE, coerce columns to schema types before validating.
stop_on_error	If TRUE, raise an error (with a summary) when the table has validation errors instead of returning them.

Value

A list with elements data (the coerced tibble) and issues (an issue table from ctdp_issues()).

Examples

```
sch <- list(name = "deployments",
  fields = list(
    list(name = "deploymentID", type = "string", constraints = list(required = TRUE)),
    list(name = "latitude", type = "number")),
  primaryKey = "deploymentID")
schema <- TableSchema$new("deployments", json = sch)
built <- ctdp_build_table(schema, data.frame(deploymentID = "A01", latitude = 35.1))
built$data
```

ctdp_check_schema *Check that a Table Schema is well-formed per the Frictionless spec*

Description

Verifies the schema structure that Frictionless requires of any Table Schema, independently of the data: non-empty `fields`, each field with a `name` and a supported `type`, constraints that are valid for the field's type, unique field names, and primary/foreign keys that reference defined fields.

Usage

```
ctdp_check_schema(x)
```

Arguments

x	A TableSchema or a parsed table-schema list.

Value

An issue table (see `ctdp_issues()`). Constraint codes: `schema-structure`, `schema-type`, `schema-constraint`, `schema-key`.

Examples

```
sch <- list(name = "deployments",
  fields = list(
    list(name = "deploymentID", type = "string", constraints = list(required = TRUE)),
   list(name = "latitude", type = "number", constraints = list(minimum = -90, maximum = 90))),
  primaryKey = "deploymentID")
ctdp_check_schema(sch)
```

ctdp_is_valid *Did a validation pass (no errors)?*

Description

Did a validation pass (no errors)?

Usage

```
ctdp_is_valid(issues)
```

Arguments

issues	An issue table.

Value

TRUE if there are no "error" severity rows.

Examples

```
ctdp_is_valid(ctdp_no_issues())   # TRUE
bad <- ctdp_issues(source = "deployments", constraint = "required",
                   severity = "error", message = "missing")
ctdp_is_valid(bad)                # FALSE
```

---

ctdp_issues *Create an issue table*

Description

Constructs a tibble of validation issues. All arguments are recycled to a common length. Use ctdp_no_issues() for an empty table with the right columns.

Usage

```
ctdp_issues(
  source,
  location_type = NA_character_,
  field = NA_character_,
  row = NA_integer_,
  constraint = NA_character_,
  severity = "error",
  message = NA_character_,
  engine = "R",
  value = NA_character_
)
```

Arguments

source	Character. Where the issue lives, e.g. "deployments.csv" or "datapackage.json".
location_type	Character. One of "table", "schema", "relation", "package".
field	Character. Column / field name, or NA.
row	Integer. 1-based data row number, or NA.
constraint	Character. The violated rule, e.g. "required", "unique", "enum", "minimum", "maximum", "minLength", "maxLength", "pattern", "type", "format", "primaryKey", "foreignKey".
severity	Character. "error" or "warning".
message	Character. Human-readable description.
engine	Character. "R" or "frictionless".
value	Character. The offending value(s), when known (e.g. the failing cell value, or the value resolved from the descriptor for a metadata error).

Value

A tibble with one row per issue.

Examples

```
ctdp_issues(source = "deployments", field = "latitude",
            constraint = "required", severity = "error",
            message = "latitude is missing")
```

`ctdp_merge_datetime` *Combine a date column and a time column into a datetime column*

Description

Generalises the original `*_date` / `*_time` merging behaviour. Produces a character column formatted in the Camtrap DP datetime format (`%Y-%m-%dT%H:%M:%S%z`) so it round-trips cleanly to CSV.

Usage

```
ctdp_merge_datetime(
  df,
  date_col,
  time_col,
  target,
  tz = "Asia/Tokyo",
  format = .ctdp_datetime_format(),
  remove = TRUE
)
```

Arguments

`df`	A data.frame / tibble.
`date_col`	Name of the date column.
`time_col`	Name of the time column.
`target`	Name of the datetime column to create.
`tz`	Time zone used to interpret the local date/time.
`format`	Output datetime format.
`remove`	If `TRUE`, drop the source date/time columns.

Value

A tibble with the `target` datetime column added.

Examples

```
raw <- data.frame(d = c("2023-04-01", "2023-04-02"), t = c("09:00:00", "10:30:00"))
ctdp_merge_datetime(raw, "d", "t", "deploymentStart")
```

ctdp_no_issues *An empty issue table*

Description

An empty issue table

Usage

```
ctdp_no_issues()
```

Value

A 0-row issue tibble.

Examples

```
ctdp_no_issues()
```

ctdp_parse_frictionless *Parse a Frictionless validation report into an issue table*

Description

Accepts either a parsed list (from jsonlite::fromJSON) or a JSON string, supporting both Frictionless v4 and v5 report layouts.

Usage

```
ctdp_parse_frictionless(report, resource_paths = NULL, descriptor = NULL)
```

Arguments

`report`	A Frictionless report as a list or JSON string.
`resource_paths`	Optional named character vector mapping resource names to file paths (e.g. `c(deployments = "deployments.csv")`) used to give a friendlier `source`. Falls back to the report's own `place/name`.
`descriptor`	Optional parsed `datapackage.json` (list). When supplied, the offending value of a **metadata** (package) error is resolved from the descriptor using the property path in the error note (e.g. `contributors[].email`) and placed in the issue's `value` column. For table/cell errors the failing `cell` value is used directly.

Value

An issue table (includes a `value` column with the offending value(s) when known).

Examples

```
# A valid Frictionless report parses to an empty issue table:
ctdp_parse_frictionless(list(valid = TRUE, tasks = list()))
## Not run:
# Typically the report comes from validate_frictionless() / the Python validator:
issues <- ctdp_validate_frictionless("path/to/package")

## End(Not run)
```

`ctdp_schema_references` *List the external (URL) references declared by a Table Schema*

Description

List the external (URL) references declared by a Table Schema

Usage

```
ctdp_schema_references(x)
```

Arguments

`x`	A TableSchema, or a parsed table-schema list (`jsonlite::fromJSON(..., simplifyVector = FALSE)`).

Value

A tibble with columns `resource`, `field` (`NA` for schema-level), `key` (the JSON key carrying the URL), `category` and `url`. Categories: `"semantic-mapping"` (skos:*), `"description-reference"`, `"example"`, `"schema-ref"` (the table schema's own URL).

Examples

```
sch <- list(name = "deployments",
  fields = list(
    list(name = "captureMethod", type = "string",
         "skos:exactMatch" = "http://rs.tdwg.org/dwc/terms/samplingProtocol")))
ctdp_schema_references(sch)
```

ctdp_semantic_only_fields

Fields whose meaning is defined only by reference (not machine-validated)

Description

Reports fields that carry a semantic mapping (`skos:*`) or a reference URL in their `description` but have **no** enforceable `enum` or `pattern` constraint. For these fields Frictionless (and this package) can check the type/format but not the controlled vocabulary, so the values should be checked against the referenced authority manually.

Usage

```
ctdp_semantic_only_fields(x)
```

Arguments

`x`	A TableSchema or a parsed table-schema list.

Value

A tibble: `resource`, `field`, `type`, `reason`, `urls`.

Examples

```
sch <- list(name = "deployments",
  fields = list(
    list(name = "captureMethod", type = "string",
         "skos:exactMatch" = "http://rs.tdwg.org/dwc/terms/samplingProtocol")))
ctdp_semantic_only_fields(sch)
```

ctdp_summarize_validation

Summarise an issue table to the console

Description

Prints a grouped summary (errors / warnings per source) followed by a detailed listing showing the offending file, field, row and message.

Usage

```
ctdp_summarize_validation(issues, max_detail = 50L)
```

Arguments

issues	An issue table from ctdp_issues().
max_detail	Maximum number of detail rows to print per source.

Value

The issue table, invisibly.

Examples

```
issues <- ctdp_issues(source = "deployments", field = "latitude",
                      constraint = "minimum", severity = "error",
                      message = "latitude below -90", value = "-100")
ctdp_summarize_validation(issues)
```

---

ctdp_validate_frictionless

Validate an existing Camtrap DP directory with Python Frictionless

---

Description

Validates a data package that already exists on disk (e.g. created by another tool or in a previous run) **without writing or overwriting** the datapackage.json or the CSV files. This is the validate-only counterpart of the R6_CamtrapDP$validate_frictionless() method, which (by default) rewrites the package from the R object before validating.

Usage

```
ctdp_validate_frictionless(
  directory,
  python = "python",
  script = NULL,
  patch_profile = TRUE,
  summarize = TRUE
)
```

Arguments

directory	Directory containing datapackage.json and its CSV files.
python	Path to the Python interpreter (with frictionless installed).
script	Path to frictionless_validate.py; resolved automatically when NULL (option, then installed inst/python/, then loose-source path).
patch_profile	If TRUE, work around the malformed $ref in the Camtrap DP 1.0 profile by validating against a locally corrected copy. This writes two **new** helper files (camtrap-dp-profile.patched.json, datapackage.validate.json) into directory only when the profile is actually malformed; it never modifies datapackage.json or the CSVs. Set FALSE to keep the directory strictly read-only.
summarize	Whether to print a summary.

Value

An issue table (see `ctdp_issues()`); engine `"frictionless"`.

Examples

```
## Not run:
# Validate an existing Camtrap DP on disk (needs Python with 'frictionless'):
issues <- ctdp_validate_frictionless("path/to/camtrapdp", python = "python")
ctdp_is_valid(issues)

## End(Not run)
```

`datapackageAdata` *Example Camtrap DP data package (acoustic)*

Description

A pre-built Camtrap DP object (class `camtrapdp`, bioacoustics flavor) created from Adep and Aobs, for trying out the acoustic workflow without rebuilding from scratch. `media` is derived from the audio file names.

Usage

```
datapackageAdata
```

Format

A `camtrapdp` object (a list with the package metadata and the `deployments` / `media` / `observations` tables under `$data`).

See Also

datapackageVdata, datapackageIdata

`datapackageIdata` *Example Camtrap DP data package (multiple camera traps)*

Description

A pre-built Camtrap DP object (class `camtrapdp`) created from Idep and Iobs, for trying out the package without rebuilding from scratch.

Usage

```
datapackageIdata
```

Format

A `camtrapdp` object (a list with the package metadata and the `deployments` / `media` / `observations` tables under `$data`).

See Also

datapackageVdata

`datapackageVdata` *Example Camtrap DP data package (single camera trap)*

Description

A pre-built Camtrap DP object (class `camtrapdp`) created from Vdep and Vobs, for trying out the package without rebuilding from scratch.

Usage

```
datapackageVdata
```

Format

A `camtrapdp` object (a list with the package metadata and the `deployments` / `media` / `observations` tables under `$data`).

References

Originally data can be used from doi:10.34462/0002000233

See Also

datapackageIdata

`Idep` *Example deployment data (multiple camera deployments with image records)*

Description

A small example deployment table used in the package vignettes and examples. One row per camera deployment.

Usage

```
Idep
```

Format

A data frame with 10 rows and 14 variables:

deploymentID Unique identifier of the deployment.

longitude Longitude in decimal degrees (WGS84).

latitude Latitude in decimal degrees (WGS84).

locationID Identifier of the deployment location.

startDate Deployment start date.

startTime Deployment start time.

endDate Deployment end date.

endTime Deployment end time.

cameraID Identifier of the camera.

cameraModel Manufacturer and model of the camera.

Delay Predefined duration after detection during which further activity is ignored (camera delay).

Height Height at which the camera was deployed.

bait Whether bait was used for the deployment.

setupBy Name or identifier of the person/organization that deployed the camera.

See Also

Iobs

Iobs *Example observation data (image records)*

Description

A small example observation table used in the package vignettes and examples. One row per observation.

Usage

```
Iobs
```

Format

A data frame with 388 rows and 17 variables:

institutionCode Institution code.

collectionCode Collection code.

obsID Observation identifier (within an event).

eventID Identifier of the event the observation belongs to.

locationID Identifier of the deployment location.

date Date the media file was recorded.

time Time the media file was recorded.

object Recorded object category (raw label).

class Taxonomic class of the observed organism.

genus Genus of the observed organism.

species Species epithet of the observed organism.

individualCount Number of observed individuals.

SDcardID Identifier of the SD card.

filename Name of the media file.

deploymentID Identifier of the deployment the observation belongs to.

eventStart Date and time at which the event started.

eventEnd Date and time at which the event ended.

See Also

Idep

`MetadataProfile` *R6 class representing a Camtrap DP package profile (metadata schema)*

Description

Frictionless validates the package *data* against the table schemas, but it validates the *metadata* (the `datapackage.json` descriptor itself) against the package **profile** – a JSON Schema (e.g. `camtrap-dp-profile.json`). `MetadataProfile` reads that profile and extracts the machine-readable metadata requirements: the **required top-level properties** (`contributors`, `project`, `spatial`, `temporal`, `taxonomic`, ...), their types / `minItems`, and the nested required keys of object properties (e.g. `project` requires `title`, `samplingDesign`, ...).

This lets the package check / scaffold the required metadata structure on the R side, instead of only discovering metadata problems when Python Frictionless runs.

Public fields

`version` Camtrap DP version.

`url` Resolved profile URL (if loaded from a URL).

`raw` The raw parsed profile (list).

`required` Character vector of required top-level property names.

`properties` Named list of top-level property definitions.

Methods

Public methods:

- MetadataProfile$new()
- MetadataProfile$property()
- MetadataProfile$property_type()
- MetadataProfile$property_min_items()
- MetadataProfile$property_required()
- MetadataProfile$item_required()
- MetadataProfile$clone()

`MetadataProfile$new()`: Create a `MetadataProfile`.

Usage:

```
MetadataProfile$new(
  version = "1.0.1",
  url = NULL,
  url_template = NULL,
  local_path = NULL,
  json = NULL,
  cache_dir = file.path(tempdir(), "camtrapdp-schemas"),
  use_cache = TRUE
)
```

Arguments:

`version` Camtrap DP version string.

`url` A fully-resolved profile URL (takes precedence over template).

`url_template` A URL containing `<version>`.

`local_path` Path to a local profile JSON file.

`json` A pre-parsed profile list.

`cache_dir` Directory used to cache downloaded profiles.

`use_cache` Reuse / store a cached copy.

`MetadataProfile$property()`: Definition of a top-level property.

Usage:

```
MetadataProfile$property(name)
```

Arguments:

`name` Property name.

`MetadataProfile$property_type()`: `type` of a property.

Usage:

```
MetadataProfile$property_type(name)
```

Arguments:

`name` Property name.

`MetadataProfile$property_min_items()`: `minItems` of a property (or `NULL`).

Usage:

```
MetadataProfile$property_min_items(name)
```

Arguments:

`name` Property name.

`MetadataProfile$property_required()`: Nested required keys of an object property (e.g. `project`).

Usage:

```
MetadataProfile$property_required(name)
```

Arguments:

`name` Property name.

`MetadataProfile$item_required()`: Required keys of each item of an array property (e.g. `taxonomic` items require `scientificName`).

Usage:

```
MetadataProfile$item_required(name)
```

Arguments:

`name` Property name.

`MetadataProfile$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MetadataProfile$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Fetches the package profile for a Camtrap DP version (needs internet):
prof <- MetadataProfile$new(version = "1.0.1")
prof

## End(Not run)
```

`R6_CamtrapDP` *R6 class representing Camtrap DP (schema-driven)*

Description

R6 class holding the Camtrap DP metadata, deployments, media and observations. This is the schema-driven successor of the original `R6_CamtrapDP`: the field names, types, constraints and relations used when adding tables are read from the Frictionless Table Schemas for the configured Camtrap DP `version`, so an arbitrary version and custom columns are handled automatically.

The public field names and the method names of the original class are preserved for backward compatibility (`set_deployments()`, `set_media()`, `set_observations()`, `set_custom()`, `add_contributors()`, `add_sources()`, `add_license()`, `set_project()`, `set_st()`, `set_taxon()`, `add_relatedIdentifiers()`, `add_references()`, `out_camtrapdp()`, `import_metadata()`). New, schema-driven capabilities are added as new methods (`get_schema()`, `add_table()`, `check_relations()`, `validate()`, `validate_frictionless()`).

Public fields

`resources` is the package data resources

`profile` of the resource

`name` Identifier of the resource

`id` A property reserved for globally unique identifiers

`created` The datetime on which this Data Package was created

`title` Title of this Data Package

`contributors` The people or organizations who contributed

`description` Description of this Data Package

`version` The version of this Data Package

`keywords` Keywords of this Data Package

`image` A URL or Path of an image for this Data Package

`homepage` A URL for the home on the web related to this Data Package

`sources` A row sources for this Data Package

`licenses` The licenses under which the Data Package is provided

`bibliographicCitation` A bibliographical reference for the resource

`project` Camera trap project or study

`coordinatePrecision` Least precise coordinate precision

`spatial` Spatial coverage, expressed as GeoJSON

`temporal` Temporal coverage of this Data Package

`taxonomic` Taxonomic coverage of this Data Package

`relatedIdentifiers` Identifiers of related resources

`references` List of references related to this Data Package

`directory` Directory of this Data Package

`data` Observation, Media and Deployments tables

`schema_urls` Named list of `<version>` URL templates per resource.

`schemas` Cache of loaded TableSchema objects, keyed by resource.

`profile_schema` Cached MetadataProfile for the package profile.

`cache_dir` Directory used to cache downloaded schemas.

`use_cache` Whether to cache / reuse downloaded schemas.

`validation` Per-table validation issues accumulated during build.

Methods

Public methods:

- `CamtrapDP$new()`
- `CamtrapDP$update_created()`
- `CamtrapDP$set_properties()`

- CamtrapDP$get_schema()
- CamtrapDP$set_deployments()
- CamtrapDP$set_media()
- CamtrapDP$set_observations()
- CamtrapDP$add_table()
- CamtrapDP$set_custom()
- CamtrapDP$add_contributors()
- CamtrapDP$add_sources()
- CamtrapDP$add_license()
- CamtrapDP$set_project()
- CamtrapDP$set_st()
- CamtrapDP$set_taxon()
- CamtrapDP$add_relatedIdentifiers()
- CamtrapDP$add_references()
- CamtrapDP$get_profile()
- CamtrapDP$metadata_requirements()
- CamtrapDP$check_metadata()
- CamtrapDP$check_descriptor()
- CamtrapDP$check_camtrap_profile()
- CamtrapDP$check_relations()
- CamtrapDP$external_references()
- CamtrapDP$validate()
- CamtrapDP$validate_frictionless()
- CamtrapDP$out_camtrapdp()
- CamtrapDP$import_metadata()
- CamtrapDP$clone()

`CamtrapDP$new()`: Creates a new instance.

Usage:

```
CamtrapDP$new(tz = "Asia/Tokyo", ...)
```

Arguments:

`tz` Time zone.

`...` Passed to `set_properties()`.

`CamtrapDP$update_created()`: Updates the created timestamp.

Usage:

```
CamtrapDP$update_created(tz = "Asia/Tokyo")
```

Arguments:

`tz` Time zone.

`CamtrapDP$set_properties()`: Sets package properties.

Usage:

```
CamtrapDP$set_properties(
  directory = getwd(),
  name = NULL,
  id = NULL,
  title = NULL,
  description = NULL,
  profile =
   "https://raw.githubusercontent.com/tdwg/camtrap-dp/<version>/camtrap-dp-profile.json",
  version = "1.0.1",
  keywords = NULL,
  image = NULL,
  homepage = NULL,
  bibliographicCitation = NULL,
  coordinatePrecision = NULL,
  schema_urls = NULL,
  cache_dir = NULL,
  use_cache = NULL
)
```

Arguments:

`directory` Directory of datapackage.

`name` Identifier of the resource.

`id` Globally unique identifier.

`title` Title of this Data Package.

`description` Description of this Data Package.

`profile` Profile of the resource (`<version>` template).

`version` The Camtrap DP version.

`keywords` Keywords.

`image` Image URL/path.

`homepage` Homepage URL.

`bibliographicCitation` Bibliographic reference.

`coordinatePrecision` Coordinate precision.

`schema_urls` Optional named list of `<version>` schema URL templates.

`cache_dir` Optional directory to cache schemas.

`use_cache` Whether to cache / reuse downloaded schemas.

`CamtrapDP$get_schema()`: Load (and cache) the TableSchema for a resource at the configured version.

Usage:

```
CamtrapDP$get_schema(
  resource,
  schema_url = NULL,
  local_path = NULL,
  json = NULL,
  refresh = FALSE
)
```

Arguments:

`resource` Resource name, e.g. `"deployments"`.

`schema_url` Optional `<version>` URL template (overrides the default).

`local_path` Optional local schema file.

`json` Optional pre-parsed schema list.

`refresh` If `TRUE`, reload even if cached on the object.

Returns: A TableSchema.

`CamtrapDP$set_deployments()`: Sets deployments. Backward-compatible signature; when the schema is reachable the data is coerced and validated against it.

Usage:

```
CamtrapDP$set_deployments(
  data,
  path = "deployments.csv",
  profile = "tabular-data-resource",
  format = "csv",
  mediatype = "text/csv",
  encoding = "utf-8",
  schema = NULL,
  mapping = NULL,
  datetime_merges = NULL,
  validate = TRUE,
  local_schema = NULL,
  tz = "Asia/Tokyo"
)
```

Arguments:

`data` Deployments dataset.

`path` Path to the data file.

`profile` Profile of the resource.

`format` Format of the data.

`mediatype` Media type.

`encoding` Encoding.

`schema` `<version>` URL template for the table schema. If `NULL` (default), the resource's configured template is used (the camera-trap default, or whatever was set via `set_properties(schema_urls=)`, e.g. a bioacoustics flavor).

`mapping` Optional column mapping (see `ctdp_apply_mapping()`).

`datetime_merges` Optional date/time merge specs.

`validate` Whether to validate against the schema.

`local_schema` Optional local schema file path.

`tz` Time zone for temporal coercion.

`CamtrapDP$set_media()`: Sets media. See `set_deployments()` for shared arguments.

Usage:

```
CamtrapDP$set_media(
  data,
  path = "media.csv",
  profile = "tabular-data-resource",
  format = "csv",
  mediatype = "text/csv",
  encoding = "utf-8",
  schema = NULL,
  mapping = NULL,
  datetime_merges = NULL,
  validate = TRUE,
  local_schema = NULL,
  tz = "Asia/Tokyo"
)
```

Arguments:

`data` Media dataset.

`path` Path to the data file.

`profile` Profile of the resource.

`format` Format of the data.

`mediatype` Media type.

`encoding` Encoding.

`schema` <version> URL template for the table schema.

`mapping` Optional column mapping.

`datetime_merges` Optional date/time merge specs.

`validate` Whether to validate against the schema.

`local_schema` Optional local schema file path.

`tz` Time zone for temporal coercion.

`CamtrapDP$set_observations()`: Sets observations. See `set_deployments()` for shared arguments.

Usage:

```
CamtrapDP$set_observations(
  data,
  path = "observations.csv",
  profile = "tabular-data-resource",
  format = "csv",
  mediatype = "text/csv",
  encoding = "utf-8",
  schema = NULL,
  mapping = NULL,
  datetime_merges = NULL,
  validate = TRUE,
  local_schema = NULL,
  tz = "Asia/Tokyo"
)
```

Arguments:

`data` Observations dataset.

`path` Path to the data file.

`profile` Profile of the resource.

`format` Format of the data.

`mediatype` Media type.

`encoding` Encoding.

`schema` <version> URL template for the table schema.

`mapping` Optional column mapping.

`datetime_merges` Optional date/time merge specs.

`validate` Whether to validate against the schema.

`local_schema` Optional local schema file path.

`tz` Time zone for temporal coercion.

`CamtrapDP$add_table()`**:** Add an arbitrary (custom or standard) resource table, schema-driven. Unlike `set_custom()` this loads a Table Schema, applies mapping, coerces, validates, stores the table in `$data[[name]]` and appends a proper tabular-data resource.

Usage:

```
CamtrapDP$add_table(
  name,
  data,
  mapping = NULL,
  datetime_merges = NULL,
  schema_url = NULL,
  local_schema = NULL,
  schema_json = NULL,
  path = NULL,
  description = NULL,
  profile = "tabular-data-resource",
  format = "csv",
  mediatype = "text/csv",
  encoding = "utf-8",
  validate = TRUE,
  tz = "Asia/Tokyo"
)
```

Arguments:

`name` Resource name.

`data` Dataset.

`mapping` Optional column mapping.

`datetime_merges` Optional date/time merge specs.

`schema_url` Optional <version> URL template for the schema.

`local_schema` Optional local schema file path.

`schema_json` Optional pre-parsed schema list.

`path` Output CSV path (defaults to <name>.csv).

`description` Optional resource description.

`profile` Resource profile.

format Resource format.
mediatype Resource media type.
encoding Resource encoding.
validate Whether to validate against the schema.
tz Time zone for temporal coercion.

`CamtrapDP$set_custom()`: Sets a custom data resource (original behaviour preserved).

Usage:

```
CamtrapDP$set_custom(name, description, data)
```

Arguments:

name Name of dataset.
description Description of dataset.
data Custom dataset.

`CamtrapDP$add_contributors()`: Adds contributors.

Usage:

```
CamtrapDP$add_contributors(contrib_table)
```

Arguments:

contrib_table data frame of contributors (title, email, path, role, organization).

`CamtrapDP$add_sources()`: Add a source.

Usage:

```
CamtrapDP$add_sources(title, path = NULL, email = NULL, version = NULL)
```

Arguments:

title Title of source.
path Path or URL to the source.
email An email address.
version The version of the source.

`CamtrapDP$add_license()`: Add a license.

Usage:

```
CamtrapDP$add_license(name, scope, path = NULL, title = NULL)
```

Arguments:

name Name of license.
scope Scope ("data" or "media").
path URL/path to the license details.
title Title of license.

`CamtrapDP$set_project()`: Sets the project.

Usage:

```
CamtrapDP$set_project(
  title,
  samplingDesign,
  captureMethod,
  individualAnimals,
  observationLevel,
  id = NULL,
  acronym = NULL,
  description = NULL,
  path = NULL
)
```

Arguments:

`title` Title of project.

`samplingDesign` Sampling design.

`captureMethod` Capture method.

`individualAnimals` Logical: individuals recognised?

`observationLevel` Observation level.

`id` Project id.

`acronym` Project acronym.

`description` Project description.

`path` Project website.

`CamtrapDP$set_st()`**:** Sets spatial and temporal coverage from the deployments.

Usage:

```
CamtrapDP$set_st()
```

`CamtrapDP$set_taxon()`**:** Sets taxonomic coverage from the observations. The Camtrap DP `taxonomic` block requires a `taxonID` (e.g. a GBIF / IUCN identifier or URI), which is looked up with `taxadb`; `taxadb` is therefore a required dependency of the package (loaded with it, as in previous versions). Names that cannot be matched get `taxonID = NA` (omitted from the output rather than a bogus `<uri>NA`). A `warning()` is emitted for `scientificName` values with unnecessary whitespace and for names with no `taxonID` in the chosen database.

Usage:

```
CamtrapDP$set_taxon(taxonDB = "gbif")
```

Arguments:

`taxonDB` Taxon database passed to taxadb: `"gbif"` (default), `"itis"` or `"ncbi"`.

`CamtrapDP$add_relatedIdentifiers()`**:** Adds a relatedIdentifier.

Usage:

```
CamtrapDP$add_relatedIdentifiers(
  relationType,
  relatedIdentifier,
  relatedIdentifierType,
  resourceTypeGeneral = NULL
)
```

Arguments:

`relationType` Type of relation.

`relatedIdentifier` Related identifier.

`relatedIdentifierType` Type of related identifier.

`resourceTypeGeneral` General type of the related resource.

`CamtrapDP$add_references()`: Adds references.

Usage:

```
CamtrapDP$add_references(reference)
```

Arguments:

`reference` Reference of data.

`CamtrapDP$get_profile()`: Load (and cache) the MetadataProfile for the package profile (the JSON Schema that Frictionless validates `datapackage.json` against). Uses `self$profile` (set by `set_properties()`), so it follows the configured version / flavor.

Usage:

```
CamtrapDP$get_profile(refresh = FALSE)
```

Arguments:

`refresh` Reload even if cached.

Returns: A MetadataProfile.

`CamtrapDP$metadata_requirements()`: The metadata requirements derived from the package profile: which top-level properties are required, their type, nested required keys, the method that creates each, and whether it is currently set.

Usage:

```
CamtrapDP$metadata_requirements()
```

Returns: A tibble (`property`, `required`, `type`, `sub_required`, `item_required`, `set_with`, `currently_set`).

`CamtrapDP$check_metadata()`: Validate the current metadata against the package profile's required structure (a fast R-side counterpart to the profile validation that Frictionless performs). Reports missing required top-level properties and missing nested/required item keys.

Usage:

```
CamtrapDP$check_metadata(summarize = TRUE)
```

Arguments:

`summarize` Print a summary.

Returns: An issue table (see `ctdp_issues()`).

`CamtrapDP$check_descriptor()`: Pre-check that the package descriptor and its table schemas conform to the Frictionless specification, before the authoritative Python Frictionless step. Checks the package structure (non-empty `resources`; each resource has `name` and `path`/`data`; unique names; `profile` set) and the well-formedness of every loaded / inline table schema (`ctdp_check_schema()`). Optionally runs a full JSON Schema validation of the written descriptor when `jsonschema` is given and the `jsonvalidate` package is installed.

Usage:

```
CamtrapDP$check_descriptor(
  check_schemas = TRUE,
  jsonschema = NULL,
  summarize = TRUE
)
```

Arguments:

`check_schemas` Also check each table schema's well-formedness.

`jsonschema` Optional path/URL to a JSON Schema to validate the serialized descriptor against (requires the `jsonvalidate` package).

`summarize` Print a summary.

Returns: An issue table (see `ctdp_issues()`).

`CamtrapDP$check_camtrap_profile()`: Warn if the package `profile` is not a Camtrap DP profile. A package can be a valid Frictionless data package yet not be Camtrap DP form unless its profile is the Camtrap DP profile.

Usage:

```
CamtrapDP$check_camtrap_profile(summarize = TRUE)
```

Arguments:

`summarize` Print a summary.

Returns: An issue table (see `ctdp_issues()`).

`CamtrapDP$check_relations()`: Check primary-key and foreign-key relations across the registered tables, driven by each table's Table Schema.

Usage:

```
CamtrapDP$check_relations(summarize = TRUE)
```

Arguments:

`summarize` If `TRUE`, print a summary of any issues.

Returns: An issue table (see `ctdp_issues()`).

`CamtrapDP$external_references()`: List every external (URL) reference declared across the package: the package `profile`, each resource's `schema` URL (or the references inside an inline schema), and the semantic / description-URL references of every loaded table schema. Use this when adopting a new version or schema flavor so that no URL-specified requirement is missed.

Usage:

```
CamtrapDP$external_references()
```

Returns: A tibble (`resource`, `field`, `key`, `category`, `url`).

`CamtrapDP$validate()`: Aggregate validation: per-table schema issues collected at build time plus cross-table relation checks. Optionally also runs the Python Frictionless validation.

Usage:

```
CamtrapDP$validate(
  relations = TRUE,
  metadata = FALSE,
  conformance = FALSE,
  frictionless = FALSE,
  summarize = TRUE,
  ...
)
```

Arguments:

`relations` Whether to run `check_relations()`.

`metadata` Whether to run `check_metadata()` (profile-driven).

`conformance` Whether to run the Frictionless conformance pre-checks `check_descriptor()` and `check_camtrap_profile()`.

`frictionless` Whether to also run `validate_frictionless()`.

`summarize` Whether to print a summary.

`...` Passed to `validate_frictionless()`.

Returns: An issue table.

`CamtrapDP$validate_frictionless()`**:** Run the Python Frictionless validation on the written data package and parse the report into an issue table.

Usage:

```
CamtrapDP$validate_frictionless(
  directory = NULL,
  python = "python",
  script = NULL,
  write = TRUE,
  patch_profile = TRUE,
  summarize = TRUE
)
```

Arguments:

`directory` Directory containing (or to receive) the data package. Defaults to a temporary directory; the package is written there first.

`python` Path to the Python interpreter.

`script` Path to `frictionless_validate.py`. If `NULL`, resolved from `getOption("camtrapdp.frictionless_script"` else the installed copy (`system.file("python/frictionless_validate.py", package = "R2camtrapdp")`), else the loose-source `python/frictionless_validate.py`.

`write` Whether to (re)write the data package before validating.

`patch_profile` If `TRUE`, work around the malformed internal `$ref` (`#$defs/version`) in the Camtrap DP 1.0 profile by validating against a locally corrected copy. The written `datapackage.json` keeps the canonical profile URL; only a separate validation descriptor is patched. Has no effect for 1.0.1 / 1.0.2 (their profiles are correct).

`summarize` Whether to print a summary.

Returns: An issue table (engine `"frictionless"`).

`CamtrapDP$out_camtrapdp()`**:** Exports the `camtrapdp` object and (optionally) writes the data package to disk.

Usage:

```
CamtrapDP$out_camtrapdp(write = FALSE, directory = NULL)
```

Arguments:

`write` If `TRUE`, write the data package to `directory`.

`directory` Output directory.

Returns: A `camtrapdp` object (list).

`CamtrapDP$import_metadata()`: Imports metadata from a list.

Usage:

```
CamtrapDP$import_metadata(metadata0)
```

Arguments:

`metadata0` List of metadata.

`CamtrapDP$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CamtrapDP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create the builder (offline):
dp <- R6_CamtrapDP$new(version = "1.0.1", title = "Example", description = "...")
## Not run:
# Registering tables fetches the schema for the chosen version (needs internet):
deployments <- create_deployments(
  deploymentID = "A01", latitude = 35.1, longitude = 139.5,
  deploymentStart_date = "2023-04-01", deploymentStart_time = "09:00:00",
  deploymentEnd_date = "2023-05-01", deploymentEnd_time = "09:00:00")
dp$set_deployments(deployments)
dp$check_relations()
dp$out_camtrapdp(write = TRUE, directory = tempfile())

## End(Not run)
```

`TableSchema` *R6 class representing a Frictionless Table Schema*

Description

`TableSchema` loads a Frictionless Table Schema (such as the Camtrap DP `deployments` / `media` / `observations` table schemas) from a URL, a local file, or an in-memory list, and exposes everything needed to build and validate a data table against it: field names and types, the primary key, foreign keys, missing-value tokens, and per-field constraints (`required`, `unique`, `enum`, `minimum`, `maximum`, `minLength`, `maxLength`, `pattern`, `type`, `format`).

Because the structure is read from the schema itself, an arbitrary Camtrap DP version and any custom / extra columns are handled automatically: any field present in the supplied schema participates in validation.

Public fields

`resource` Resource name, e.g. `"deployments"`.

`version` Camtrap DP version used to resolve the URL.

`url` Resolved schema URL (if loaded from a URL).

`name` Schema `name`.

`title` Schema `title`.

`description` Schema `description`.

`fields` Named list of field definitions keyed by field name.

`field_order` Character vector of field names, in schema order.

`primaryKey` Character vector naming the primary key field(s).

`foreignKeys` List of foreign-key definitions.

`missingValues` Character vector of missing-value tokens.

`raw` The raw parsed schema (list).

Methods

Public methods:

- `TableSchema$new()`
- `TableSchema$field_names()`
- `TableSchema$field()`
- `TableSchema$required_field_names()`
- `TableSchema$field_type()`
- `TableSchema$requirements()`
- `TableSchema$empty_table()`
- `TableSchema$coerce()`
- `TableSchema$external_references()`
- `TableSchema$semantic_only_fields()`
- `TableSchema$check_schema()`
- `TableSchema$validate()`
- `TableSchema$clone()`

`TableSchema$new()`: Create a `TableSchema`.

Usage:

```
TableSchema$new(
  resource = NULL,
  version = "1.0.1",
  url_template = NULL,
  local_path = NULL,
  json = NULL,
  cache_dir = file.path(tempdir(), "camtrapdp-schemas"),
  use_cache = TRUE
)
```

Arguments:

`resource` Resource name (used to pick a default URL template and to label issues), e.g. `"deployments"`.

`version` Camtrap DP version string, e.g. `"1.0.1"`.

`url_template` URL containing the `<version>` placeholder. If `NULL` and `resource` is one of the standard tables, a default template is used.

`local_path` Path to a local schema JSON file. Takes precedence over the URL.

`json` A pre-parsed schema list. Takes precedence over `local_path`.

`cache_dir` Directory used to cache downloaded schemas.

`use_cache` If `TRUE`, reuse a cached copy when present and cache new downloads.

`TableSchema$field_names()`: Field names in schema order.

Usage:

```
TableSchema$field_names()
```

`TableSchema$field()`: Get a single field definition by name.

Usage:

```
TableSchema$field(name)
```

Arguments:

`name` Field name.

`TableSchema$required_field_names()`: Names of fields whose `constraints$required` is `TRUE`.

Usage:

```
TableSchema$required_field_names()
```

`TableSchema$field_type()`: Type of a field (`"string"`, `"number"`, ...).

Usage:

```
TableSchema$field_type(name)
```

Arguments:

`name` Field name.

`TableSchema$requirements()`: A tidy summary of the schema's requirements: one row per field with its `type`, `format`, and constraints (`required`, `unique`, `enum`, `minimum`, `maximum`, `pattern`). Works for any version / flavor.

Usage:

```
TableSchema$requirements()
```

Returns: A tibble.

`TableSchema$empty_table()`: Create an empty (0-row) tibble shell with one correctly typed column per schema field, in schema order.

Usage:

```
TableSchema$empty_table()
```

Returns: A tibble with 0 rows.

`TableSchema$coerce()`: Coerce a data frame to the schema. The result contains **every** schema field, in schema order: present columns are cast to the schema type (date/datetime/time formatted as canonical strings) and fields absent from the input are added as typed `NA` columns (Camtrap DP CSVs are expected to carry all schema columns). Columns not present in the schema are kept after the schema columns (custom columns) and a warning is emitted listing them.

Usage:

```
TableSchema$coerce(df, tz = "Asia/Tokyo", complete = TRUE)
```

Arguments:

`df` A data.frame / tibble.

`tz` Time zone used when formatting date/datetime values supplied as `POSIXt` / `Date`.

`complete` If `TRUE` (default), include all schema fields, filling absent ones with typed `NA`. If `FALSE`, keep only the supplied columns.

Returns: A tibble.

`TableSchema$external_references()`: List the external (URL) references this schema declares (semantic `skos:*` mappings, reference URLs in field descriptions, the schema URL itself). See `ctdp_schema_references()`.

Usage:

```
TableSchema$external_references()
```

`TableSchema$semantic_only_fields()`: List fields whose meaning is defined only by reference (a semantic mapping or a description URL) and which therefore cannot be fully validated against a controlled vocabulary. See `ctdp_semantic_only_fields()`.

Usage:

```
TableSchema$semantic_only_fields()
```

`TableSchema$check_schema()`: Check that this schema is a well-formed Frictionless Table Schema (supported types, constraints valid for each type, keys reference defined fields). See `ctdp_check_schema()`.

Usage:

```
TableSchema$check_schema()
```

`TableSchema$validate()`: Validate a data frame against the schema constraints.

Usage:

```
TableSchema$validate(
  df,
  source = paste0(self$resource %||% self$name, ".csv"),
  raw = NULL
)
```

Arguments:

`df` A data.frame / tibble (ideally already passed through `$coerce()`).

`source` Label used as the issue `source` (e.g. `"deployments.csv"`).

`raw` Optional pre-coercion data (the original input passed to `$coerce()`). When supplied, values that were present in `raw` but became `NA` during coercion – i.e. type-invalid entries such as a non-numeric string in a `number` field – are reported as `type` errors instead of silently vanishing. `ctdp_build_table()` passes this automatically.

Returns: An issue table (see `ctdp_issues()`).

`TableSchema$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TableSchema$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Build from an in-memory schema (offline):
sch <- list(name = "deployments",
  fields = list(
    list(name = "deploymentID", type = "string", constraints = list(required = TRUE)),
    list(name = "latitude", type = "number")),
  primaryKey = "deploymentID")
schema <- TableSchema$new("deployments", json = sch)
schema$field_names()
schema$required_field_names()
## Not run:
# Or fetch the official schema for a Camtrap DP version (needs internet):
TableSchema$new("deployments", version = "1.0.1")

## End(Not run)
```

Vdep *Example single camera-trap deployment data (video)*

Description

Example deployment data for a single camera trap (at NIES, Japan), used in the single-camera vignette. One row.

Usage

```
Vdep
```

Format

A data frame with 1 row and 14 variables:

deploymentID Unique identifier of the deployment.

longitude Longitude in decimal degrees (WGS84).

latitude Latitude in decimal degrees (WGS84).

locationID Identifier of the deployment location.

startDate Deployment start date.

startTime Deployment start time.

endDate Deployment end date.

endTime Deployment end time.

cameraID Identifier of the camera.

cameraModel Manufacturer and model of the camera.

Delay Camera delay.

Height Height at which the camera was deployed.

bait Whether bait was used.

setupBy Name or identifier of the person/organization that deployed the camera.

References

Originally data can be used from doi:10.34462/0002000233

See Also

Vobs

`Vobs` *Example single camera-trap observation data (video)*

Description

Example observation data for the Vdep single camera-trap deployment, used in the single-camera vignette. One row per observation; `filename` is the video file from which the `media` table is built.

Usage

```
Vobs
```

Format

A data frame with 38 rows and 13 variables:

institutionCode Institution code.

collectionCode Collection code.

videoID Video identifier.

locationID Identifier of the deployment location.

date Date the video was recorded.

time Time the video was recorded.

object Recorded object category.

class Taxonomic class of the observed organism.

genus Genus of the observed organism.

species Species epithet of the observed organism.

individualCount Number of observed individuals.

SDcardID Identifier of the SD card.

filename Name of the video file (used to build `media`).

References

Originally data can be used from doi:10.34462/0002000233

See Also

Vdep

Index

∗ **datasets**
  Adep, 3
  Aobs, 4
  datapackageAdata, 22
  datapackageIdata, 22
  datapackageVdata, 23
  Idep, 23
  Iobs, 24
  Vdep, 43
  Vobs, 44

Adep, 3, *4*, *22*
Aobs, *3*, 4, *22*

create_deployments, 5
create_media, 7
create_observations, 8
ctdp-build-table, 11
ctdp-conformance, 11
ctdp-frictionless, 11
ctdp-mapping, 11
ctdp-references, 12
ctdp-validation-report, 12
ctdp_apply_mapping, 12
ctdp_apply_mapping(), *14*, *31*
ctdp_bind_issues, 13
ctdp_build_table, 13
ctdp_build_table(), *42*
ctdp_check_schema, 15
ctdp_check_schema(), *42*
ctdp_is_valid, 15
ctdp_issues, 16
ctdp_issues(), *12*, *14*, *15*, *21*, *22*, *36*, *37*, *43*
ctdp_merge_datetime, 17
ctdp_merge_datetime(), *14*
ctdp_no_issues, 18
ctdp_no_issues(), *16*
ctdp_parse_frictionless, 18
ctdp_schema_references, 19
ctdp_schema_references(), *42*
ctdp_semantic_only_fields, 20
ctdp_semantic_only_fields(), *42*
ctdp_summarize_validation, 20
ctdp_validate_frictionless, 21
ctdp_validate_frictionless(), *11*

datapackageAdata, 22
datapackageIdata, 22, *22*, *23*
datapackageVdata, *22*, 23, *23*

Idep, *22*, 23, *25*
Iobs, *22*, 24, *24*

jsonlite::fromJSON, *18*

MetadataProfile, 25, *28*, *36*

R6_CamtrapDP, *11*, 27

TableSchema, *11*, *14*, *15*, *19*, *20*, *28*, *30*, *31*, 39
tibble::tibble, *12*

Vdep, *23*, 43, *44*, *45*
Vobs, *23*, 44, *44*