Package: camtrapReport (via r-universe)

November 9, 2026

Type Package

Title Camera-Trap Report Generator

Version 1.0.73

Description Provides an extensible workflow for processing camera-trap data in the 'Camtrap DP' format and automatically generating reproducible data-status and ecological reports in HTML or PDF format. The package supports data-quality checks, ecological summaries, visualisations, maps, tables and modular report sections for wildlife-monitoring applications. Optional taxonomic enrichment queries the Global Biodiversity Information Facility Species API <https://techdocs.gbif.org/en/openapi/v1/species> through 'taxize'. Browser-based map backgrounds use 'OpenStreetMap' <https://www.openstreetmap.org>, 'OSM Germany' <https://openstreetmap.de/germanstyle/> and 'OpenTopoMap' <https://opentopomap.org>, subject to their usage terms.

License MIT + file LICENSE

URL https://spatialecology.github.io/camtrapReport/, https://github.com/spatialecology/camtrapReport, https://camera-traps.r-universe.dev/camtrapReport

BugReports https://github.com/spatialecology/camtrapReport/issues

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), dplyr (>= 1.1.1), shiny (>= 1.7.0), glue (>= 1.6.0), jsonlite (>= 1.8.0), lubridate (>= 1.9.0), knitr (>= 1.40), methods, rmarkdown (>= 2.20), terra (>= 1.8-10)

Suggests roxygen2, pak, pagedown, testthat (>= 3.1.7), xml2, taxize, lutz, spatstat, spatstat.geom, spatstat.explore, camtrapDensity, suncalc, activity, here

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

VignetteBuilder knitr

SystemRequirements Google Chrome or Microsoft Edge (optional, for PDF output)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs
cmake libgdal-dev gdal-bin libgeos-dev make libuv1-dev libproj-dev libsqlite3-dev zlib1g-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-11-09 12:53:24 UTC

RemoteUrl https://github.com/spatialecology/camtrapReport

RemoteRef HEAD

RemoteSha 9414281703d213606811d19cbd5f3768f9169d47

Contents

camData,character-method 2
camReport-classes 6
gui,camReport-method 8
info,camReport-method 9
install_all 11
modules 13
report 15
reportProfile 18
reportSection,character-method 19
section_names 21
testSection,.textSection-method 22
updateReportSection 24

Index **27**

`camData,character-method`
Read camera-trap data in Camtrap DP format

Description

Create a `camReport` object from a Camtrap DP dataset.

Usage

```
camData(data, habitat, study_area, update, enrich_taxonomy = FALSE,
  ..., overwrite = FALSE)
```

Arguments

data	A character string giving the path to a Camtrap DP dataset, provided as a ZIP file or an extracted dataset directory.
habitat	An optional data frame containing habitat information for camera locations. The default is NULL.
study_area	An optional study-area boundary provided as a spatial file path, a SpatVector, or an sf object. The default is NULL.
update	A logical value (default FALSE) specifying whether to recreate a previously saved camReport object.
enrich_taxonomy	A single logical value (default FALSE) allowing GBIF taxonomy enrichment when the optional taxize package is installed. Only explicit TRUE permits these network requests. This affects fresh imports; use update = TRUE to rebuild a previously cached object.
...	Named persistence settings: path is an existing parent directory for ZIP extraction (required for ZIP inputs); cache_file is an explicit .rds filename for saving or reusing the processed object. Both default to NULL. Other arguments remain reserved. No positional arguments are added.
overwrite	A single logical value (default FALSE) allowing a ZIP input to replace archived files in an existing extraction directory. Also permits replacing an explicit cache_file when rebuilding. Other files in that directory are retained. This does not control reuse of a cached object when data is already a dataset directory; see update.

Details

The function reads the input Camtrap DP dataset and creates a camReport object. The resulting object contains processed camera-trap data, metadata, summaries, report text, report modules, and data-status information used to generate automated reports.

If habitat information is provided, it is linked to camera locations and used in habitat-related summaries. If a study-area boundary is provided, it is stored with the object and used in spatial summaries where relevant.

Imports do not save a cache unless cache_file is supplied through .... Existing __camReport_Object.rds directory caches are still reused unless update = TRUE; they are never overwritten implicitly.

data currently accepts a file-system path, not an in-memory data frame or list. A dataset directory contains datapackage.json, deployments.csv, media.csv, and observations.csv. The following identifiers and columns explain the main relationships; this is not a complete Camtrap DP schema:

File	Contents and important links
datapackage.json	Project metadata and a taxonomic list with species names, taxon identifiers, ranks, families, orders, and
deployments.csv	One camera deployment per deploymentID, with locationID, locationName, longitude, latitude,
media.csv	Images or videos identified by mediaID, linked to deploymentID, with timestamp and optional capture
observations.csv	Species or other classifications linked to deploymentID, mediaID and/or eventID, with scientificNa

A deployment is a period during which a camera is installed at a location. An observation records a classification at media or event level. An event groups related media; camtrapReport uses `sequenceID` internally for this event grouping and derives a `sequences` table from media timestamps. These internal sequences are not an additional input CSV file. See the Camtrap DP specification for definitions and the example-data provenance for a concrete dataset.

Import and `status()` provide package-specific data checks, not formal validation against the complete Camtrap DP schema. Check schema compliance upstream. After correcting source CSV or JSON files, call `camData()` again with `update = TRUE` and the optional inputs, then regenerate `status()`. Merely rerunning `status()` on the old object does not reread changed files.

Value

A `camReport` object.

Files and cache reuse

Folder imports do not write beside the source data. Study areas remain in memory; existing `study_area.map` files are left untouched. ZIP imports require an explicit `path`, extracting into a subdirectory named after the archive. Choose persistent storage when retaining objects across sessions. Existing extraction targets require `overwrite = TRUE`; unrelated files remain and extraction is not transactional. Package installation directories are rejected.

Use `camData(data_dir, cache_file = file.path(output_dir, "analysis.rds"))` to save an object explicitly. An existing cache is reused unless `update = TRUE`; rebuilding also requires `overwrite = TRUE`. Study areas are packed into the saved object and restored by `camData(data_dir, cache_file = ...)`. The GUI's Save object action also packs study areas. Keep the source dataset at its recorded location for image access and source freshness checks. Legacy directory caches remain readable when no `cache_file` is supplied. To replace one explicitly, supply its full filename as `cache_file`, together with `update = TRUE, overwrite = TRUE`. Lightweight file-size and modification-time checks warn if the four source CSV/JSON files were changed, added or removed in the dataset root or their recorded directories. The cache is still reused; use `update = TRUE` to rebuild it. Older caches without this metadata also warn because their freshness cannot be verified. No input contents are hashed or reread during this check. Changes that preserve both size and modification time, files in previously unknown nested directories, and changes to optional `habitat` or `study_area` inputs are not detected; rebuild explicitly for these. Only load datasets and saved R objects from sources you trust.

ZIP inventories are checked before extraction for unsafe or ambiguous paths and resource limits. The defaults are 100,000 entries (including directories) and 20 GiB of total declared uncompressed size. To adjust these limits, set `options(camtrapReport.max_zip_entries = 100000, camtrapReport.max_zip_bytes = 20 * 1024^3)` before import. Each limit must be a positive finite number. These checks are practical safeguards, not a security sandbox or complete protection against malicious archives.

Optional taxonomy enrichment

The default `enrich_taxonomy = FALSE` never initiates GBIF taxonomy requests, whether or not `taxize` is installed. With explicit `enrich_taxonomy = TRUE` and `taxize` installed, a fresh import sends scientific names to GBIF to fill missing class and order information. A message announces this network access. Requests can affect import time, privacy and reproducibility. Source-provided

names, identifiers and valid classifications are retained; external results fill only missing values after an unambiguous identifier and name match. Existing normalization of taxon-ID URLs to their last path component is unchanged; the original metadata remain available in `object$info$json`. Ambiguous, unmatched or incomplete results remain missing, with a concise warning that does not list the species inventory. Optional lookup failures do not prevent import of the underlying data.

No package-level timeout or complete-operation time limit is imposed. The supported `taxize` lookup stages do not offer one consistently verified end-to-end timeout interface; R's global `timeout` option is not a guarantee for these HTTP requests. For predictable offline operation, leave enrichment disabled and supply adequate taxonomy in `datapackage.json`.

Without enrichment, imported species names, families, orders and counts are retained, including classes when supplied in the source. Taxonomic classes may otherwise be unavailable. Groups defined by class (for example `wild_mammals`, birds, reptiles or amphibians) may then be absent. Setup reports missing classifications; existing mammal-count summaries can be incomplete and must not be interpreted as ecological absence. Explicit class-based capture requests require available class metadata with a verified match when classification is incomplete; otherwise they stop with a clear error. Partial metadata with verified matches produces a warning. Default order-based groups remain usable when the source supplies the relevant orders. This option does not remove taxonomy already present in a reused cache. Reusing `__camReport_Object.rds` does not query GBIF, even if the enrichment setting changes; use `update = TRUE` to rebuild with that setting. This is a processed-object cache, not a cache of individual GBIF responses. Remote taxonomy can change between fresh lookups. Preserve the source data, processed object and package versions used in analyses.

Study areas that change over time

`study_area` accepts one boundary per object. Multiple Shiny uploads are the component files of a shapefile (`.shp`, `.shx`, `.dbf`, `.prj`, etc.), not a set of year-specific boundaries. For different survey footprints, prepare separate year-specific datasets and objects with the corresponding boundary, or supply a custom module with an explicit year-to-boundary mapping.

See Also

`report()`, `status()`, `info()`, `gui()`

Examples

```
# Small import demonstration, not data for ecological inference.
# The larger analytical example is in system.file("external", "dataset",
# package = "camtrapReport"); see the vignettes for its complete workflow.
source_dataset <- system.file(
  "external",
  "example-small",
  package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
))
```

```
cm <- camData(example_dataset)

cm

unlink(example_dataset, recursive = TRUE, force = TRUE)
```

`camReport-classes` *camReport classes*

Description

Class definitions used by camtrapReport to store camera-trap data, report metadata, report sections, and intermediate report objects.

Value

This is a documentation-only topic and does not return a value.

camReport class

The `camReport` class is the main object class used by camtrapReport. Objects of this class are usually created with `camData()`. A `camReport` object stores the camera-trap dataset, report metadata, processed summaries, report modules, status-report modules, and settings used for report generation.

The internal `camR` object is the low-level Reference Class generator for advanced use. Calling `camR$new()` creates an unprepared container; it does not import data or run the setup and data-status checks performed by `camData()`. Use `camData()` for the normal reporting workflow.

Main fields

Important fields in a `camReport` object include:

- `data`: A list containing the camera-trap data tables, including observations, deployments, media, locations, sequences, and taxonomy.
- `habitat`: Optional habitat information linked to camera locations.
- `study_area`: Optional spatial object or path describing the study area.
- `siteName`: Name of the study site.
- `title`: Title used in the generated report.
- `subtitle`: Subtitle used in the generated report.
- `authors`: Author or contributor text used in the generated report.
- `institute`: Institute or organisation text used in the generated report.
- `description`: Study-area or project description used in the generated report.
- `years`: Years included in the report summaries.

- group_definition: Definitions of species or observation groups used in the report.
- setting: Report settings, including selected focus groups, rem_parameter_scope ("pooled" or "annual"), rem_reps, rem_seed, and map_basemap ("offline", "cartodb", "esri", or "openstreetmap", the default). For noncommercial research reports, the OpenStreetMap mode tries OpenStreetMap Standard, OpenStreetMap Germany, and OpenTopoMap in that order, then a neutral background if all fail. There is no additional eligibility setting. The services' usage terms and visible attributions still apply; no online service guarantees availability. Local file:// reports skip Standard because they cannot send its required web Referer, and start with Germany. Serve the report over HTTP(S) to try Standard first. No header is fabricated. "offline" makes no tile requests. Existing explicit CARTO and Esri settings remain subject to their operators' applicable terms. Bundled report modules select their REM scope explicitly; the standard profile is pooled and the EOW profile is annual.
- data_status: Data-status summaries generated from the input dataset.
- reportObjectElements: Report modules and related objects used to generate the ecological report.
- statusReportObjects: Objects used to generate the data-status report.

Time zones and year boundaries

Imported timestamps without an explicit offset are parsed as UTC. Setup uses UTC only if no explicit, existing or data-provided time-zone setting is available. A call such as cm$setup(tz = "UTC") does not reparse or shift stored timestamps; existing cached settings are otherwise preserved.

Deployment subsets use the deployment's start year, including linked records after New Year. A year with no matching deployments raises an explicit error. Some annual summaries instead group observations by timestamp calendar year. These conventions may differ for deployments spanning New Year; the package does not automatically split effort or reallocate observations at the boundary. Review those cases before comparing annual results.

Supporting classes

camtrapReport also defines supporting classes and class unions used internally, including camInfo, .Rchunk, .textSection, characterORnull, characterORlist, characterORlistORnull, listORnull, and data.frameORnull.

See Also

camData(), report(), status(), info(), reportSection()

Examples

```
# Inspect the formal definition of the main report class.
methods::getClass("camReport")

# Check whether the class is registered.
methods::isClass("camReport")
```

`gui,camReport-method` *Launch the camtrapReport GUI*

Description

Launch an interactive Shiny interface for configuring and generating camtrapReport outputs from an existing `camReport` object.

Usage

```
gui(object, launch.browser, max_upload_mb, ...)
```

Arguments

`object`	A `camReport` object, usually created with `camData()`.
`launch.browser`	A logical value (default `TRUE`) indicating whether the application is opened in the default web browser.
`max_upload_mb`	A numeric value (default `2000`) giving the maximum file-upload size in megabytes.
`...`	Additional arguments passed to `shiny::runApp()`.

Details

The function starts a Shiny-based graphical user interface for editing report metadata, selecting report sections, defining focus groups, and generating data-status or ecological reports.

The GUI can be used with an existing `camReport` object and provides an interactive alternative to configuring reports directly in R. Choose an existing output directory explicitly before loading ZIP data or saving output. ZIP data are extracted there so saved objects retain their source paths across R sessions. Existing report, R Markdown, and saved-object files are protected unless the overwrite checkbox is selected. Generating output does not change the R working directory. On local Windows sessions, Browse opens R's native folder chooser; manual path entry remains available on every platform. The Sections tab edits the loaded source narrative without evaluating its placeholders or replacing code-generated content. Reset restores the text present when the dataset was loaded into the GUI. Save the current object as RDS to retain edits to included sections. Excluded sections retain their edits only for the current GUI session. Focus-group choices come from the data; missing taxonomy is explained without automatically contacting GBIF. ZIP imports have a separate, initially unchecked permission to overwrite an existing extraction directory; see `camData()` for its location and limits. GBIF taxonomy enrichment is also initially unchecked. Selecting it permits sending scientific names to GBIF when importing fresh data with `taxize` installed. Loading an existing processed-object cache does not enrich it.

Value

Invisibly returns the supplied `camReport` object after launching the application.

See Also

camData(), report(), status(), info()

Other report generation: report()

Examples

```
# Set CAMTRAPREPORT_RUN_GUI_EXAMPLE=true deliberately to launch this example.
source_dataset <- system.file(
  "external",
  "dataset",
  package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
))

cm <- camData(example_dataset)

gui(cm)

unlink(example_dataset, recursive = TRUE, force = TRUE)
```

---

info,camReport-method *Get or set information in a camReport object*

Description

Get selected information fields from a camReport object or update the value of a field used in report generation.

Usage

```
## S4 method for signature 'camReport'
info(x, name)

## S4 replacement method for signature 'camReport'
info(x, name) <- value
```

Arguments

x	A camReport object created by camData().

name	A character vector naming fields to retrieve, or a single character string naming the field to update. When omitted from info(), the default report-information fields are returned.
value	The new value to assign to the specified field.

Details

A camReport object contains metadata and report information extracted from the camera-trap dataset, such as title, subtitle, authors, institute, site name, description, and acknowledgement text.

The info function can be used to inspect selected fields. The replacement form, info(x, name) <- value, can be used to update report information before generating the report.

The getter accepts actual camReport field names, such as "title" and "description". The replacement form also recognises the following convenience names (case-insensitive):

Replacement name	Updated content
"study area"	The description field.
"introduction"	Text of the introduction module.
"image processing"	Text of the image-processing module.
"sampling effort" or "sampling efforts"	Text of the sampling-effort module.
"acknowledgement" or "acknowledgements"	Text of the acknowledgements module.

Other replacement names must match a field name exactly. To target a particular module variant, use updateReportSection() with its module name. Edits to report text affect the object; they do not rewrite the original dataset or the bundled YAML files.

Value

info() returns an object of class camInfo containing the requested fields. The replacement method returns the updated camReport object invisibly.

See Also

camData(), report(), status(), updateReportSection()

Examples

```
# Use the small dataset to demonstrate metadata access and editing.
source_dataset <- system.file(
  "external",
  "example-small",
  package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
```

```
))

cm <- camData(example_dataset)

# Retrieve all report information
info(cm)

# Retrieve selected fields
selected_info <- info(
  cm,
  name = c("title", "authors", "institute")
)

selected_info

# Update individual fields
info(cm, "title") <- "Camera-trap monitoring report"
info(cm, "institute") <- "Example Wildlife Research Institute"

# Inspect the updated fields
info(
  cm,
  name = c("title", "institute")
)

unlink(example_dataset, recursive = TRUE, force = TRUE)
```

`install_all` *Install packages required by camtrapReport*

Description

Install packages in camtrapReport's module inventory and additional packages requested through `pkgs`.

Usage

```
install_all(pkgs = NULL, update = FALSE, github = TRUE,
  gitlab = FALSE, ...)

## S4 method for signature 'ANY'
install_all(pkgs = NULL, update = FALSE, github = TRUE, gitlab = FALSE, ...)
```

Arguments

`pkgs`	An optional character vector of additional package names. The default is `NULL`.
`update`	A logical value. If `TRUE`, optional packages are force-reinstalled. The default is `FALSE`.

github	A logical value (default TRUE) specifying whether packages configured in GitHub repositories are included.
gitlab	A logical value (default FALSE) specifying whether packages configured in GitLab repositories are included.
...	Additional named arguments passed to pak::pkg_install(), such as lib, ask, dependencies, or upgrade.

Details

The function checks the package list used by camtrapReport and installs packages that are not currently available. Packages configured as GitHub or GitLab dependencies are installed from their corresponding repositories. All package installation is handled by pak::pkg_install().

Unlike a direct call to pak, install_all() discovers package declarations in the installed report-module pool and any packaged configuration inventory. It does not inspect a project's copied module registry or an object's selected profile. Supply custom-module and other optional analytical requirements explicitly through pkgs; these are added to, not substituted for, the discovered inventory. It then delegates resolution and installation to pak. The function is opt-in and is never called when the package is loaded or a report is rendered.

install_All() is retained as a deprecated compatibility alias. New code should use install_all().

To inspect optional packages without installing anything, examine module YAML #| packages: declarations (see the module guide). Use pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)] to check a character vector of package names. There is currently no dry_run argument; passing that name through ... is not a supported inspection API.

When update = TRUE, the requested optional packages are force-reinstalled from their configured sources using the pak reinstall parameter. Dependencies are upgraded only when required by the requested packages, unless upgrade = TRUE is supplied through ....

Value

Called for its side effects. The function returns NULL invisibly.

See Also

camData(), report(), status()

Examples

```
# Inspect one module's optional packages without installing anything.
module_file <- system.file(
  "reportSections", "sampling.yml", package = "camtrapReport"
)
declaration <- grep(
  "#| packages:", readLines(module_file, warn = FALSE),
  value = TRUE, fixed = TRUE
)
pkgs <- trimws(strsplit(
  sub("#| packages:", "", declaration, fixed = TRUE), ","
)[[1]])
```

```
pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]

# Package installation changes the user's R library, so these examples are
# enabled only by a deliberate environment-variable opt-in.
if (interactive() && identical(Sys.getenv("CAMTRAPREPORT_INSTALL_EXAMPLE"), "true")) {
install_all()
install_all(pkgs = "readr")
install_all(update = TRUE)

# Reinstall the requested packages and also update their dependencies.
install_all(update = TRUE, upgrade = TRUE)
}
```

`modules` *Manage report modules*

Description

Security: Modules can execute arbitrary R code with the permissions of your R process during testing or report generation. Add and run only modules from sources you trust, and inspect their code, package declarations, network requests and file operations first. Modules are not sandboxed.

Usage

```
add_Module(x, before, after, test, object, formats, dir)

move_Module(name, before, after, parent, level0, formats, dir)

remove_Module(name, recursive, dir)

restore_Module(name, batch_id, test, dir)

empty_trash(name, id, dir)

list_Modules(tree, brief, include_trash, validate, dir)
```

Arguments

`x`	A character string giving the path to a module YAML file to add to the package.
`before`	An optional character string naming the module before which the new or moved module should be placed. The default is `NULL`.
`after`	An optional character string naming the module after which the new or moved module should be placed. The default is `NULL`.
`test`	A logical value specifying whether a module is tested before being added or restored. For `add_Module()`, the default is `FALSE` when `object` is not supplied and `TRUE` when a `camReport` object is supplied. For `restore_Module()`, the default is `TRUE`.

object	An optional camReport object used when testing a module that requires access to report data. The default is NULL.
formats	Output compatibility for the module: "html", "pdf", or "both". For add_Module(), an omitted value is read from an optional formats field in the YAML file and otherwise defaults to "both". For move_Module(), an omitted value retains the current setting.
dir	Path to a writable module registry containing __modulesList.csv. This argument is required for modifying modules. The registry bundled with the package is read-only and cannot be modified directly.
name	A character string naming the module to move, remove, restore, or permanently delete from the trash. For empty_trash(), the default is NULL.
parent	An optional character string naming the parent module. If omitted, the current parent is retained when moving a module.
level0	An optional character vector specifying the top-level module order. The default is c("introduction", "methods", "results", "acknowledgements", "appendix").
recursive	A logical value (default TRUE) specifying whether child modules are removed together with the selected parent module.
batch_id	An optional character string identifying a deleted-module batch to restore. The default is NULL.
id	An optional character string identifying a deleted-module batch to remove permanently from the trash. The default is NULL.
tree	A logical value (default TRUE). If TRUE, modules are returned as a hierarchical tree.
brief	A logical value (default TRUE). If TRUE, a shortened module summary is returned when tree = FALSE.
include_trash	A logical value (default FALSE) specifying whether deleted modules in the trash are also listed.
validate	A logical value (default FALSE) specifying whether module YAML files are checked for validity when listing modules.

Details

Manage report modules in camtrapReport. These functions can be used to list existing modules, add new modules, move modules within the report structure, remove modules, restore deleted modules, and empty the module trash.

camtrapReport uses a modular report architecture in which each report section is stored as a module. A module can contain section metadata, text, executable R code, required packages, and rendering settings.

add_Module() adds a new YAML-defined module to a module registry. The module can optionally be inserted before or after an existing module.

move_Module() changes the position or parent of an existing module within the hierarchical report structure.

`remove_Module()` moves a module to the module trash. If `recursive = TRUE`, child modules are also moved to the trash.

`restore_Module()` restores a previously removed module from the trash.

`empty_trash()` permanently removes modules from the trash.

`list_Modules()` lists active modules and can optionally include deleted modules stored in the trash.

Supply `dir` when changing modules. If it is omitted, the bundled registry is available for reading only. Copy `reportSections` to a writable project directory first, then pass that copied directory to `add_Module()` and the other management functions. See the module guide.

YAML modules contain executable R code. Add and run only modules from trusted sources; inspect their code, package declarations, network requests, and file operations before running them. A profile selects modules but does not sandbox their execution.

Value

`list_Modules()` returns a data frame, a hierarchical module tree, or a list containing active and trashed modules. The other functions return information about the performed operation invisibly.

See Also

`reportSection()`, `updateReportSection()`, `testSection()`, `report()`

Examples

```
# List modules as a hierarchical tree
module_tree <- list_Modules()
head(module_tree)

# List modules in table form
module_table <- list_Modules(tree = FALSE)
head(module_table)
```

---

report *Generate ecological and data-status reports*

Description

Generate reports from a `camReport` object. `report()` creates an ecological report, while `status()` creates a data-status report. A named `profile` can select an ordered set of modules immediately before rendering.

Usage

```
report(object, filename, view, test, profile, format, overwrite = FALSE)

status(object, filename, view, profile, format, overwrite = FALSE)
```

Arguments

object	A camReport object created by camData().
filename	An optional output filename or path. A .html or .pdf extension selects the format when format is omitted. Without an extension, .html is appended. Omitted filenames use temporary output.
view	A logical value (default FALSE) specifying whether to open the generated report.
test	A logical value (default FALSE). If TRUE, ecological report modules are tested when HTML rendering fails.
profile	An optional profile name or reportProfile object. When omitted, the sections currently attached to object are rendered.
format	Output format, either "html" or "pdf". The default is inferred from filename, falling back to "html".
overwrite	A logical value (default FALSE). Set to TRUE explicitly to replace existing output and .Rmd files with the selected stem.

Details

HTML is the default output. PDF output first renders the existing HTML report and then prints it through a headless Chrome or Edge browser using the optional pagedown package. This route preserves modules that contain HTML widgets or HTML-formatted tables. Interactive controls are naturally static in the resulting PDF. PDF output omits the table of contents and excludes modules whose registry entry does not support PDF. The formats column shown by list_Modules() records module compatibility.

Output includes the rendered .html or .pdf file and an .Rmd source file with the same stem. The resolved paths are displayed before rendering. An omitted or empty filename uses a unique R-session temporary file. An explicitly supplied filename without a directory uses the object's existing dataset directory; otherwise supply a filename including an existing output directory. Invalid directories cause an error rather than falling back to a different location. Existing report or .Rmd files are protected unless overwrite = TRUE. Supply a new filename for each retained report; use tempfile() for a preview.

Selected module packages are checked with requireNamespace() without automatically attaching them. Bundled modules qualify external functions; custom modules that relied on automatic attachment must use package::fun() or otherwise manage their own dependencies explicitly. Module code can still change the search path or other process-wide state. The rendering environment is not a separate R process or a sandbox. Run only modules from sources you trust.

view = TRUE opens the completed file after rendering; it does not select different modules or dependencies. To inspect a file that was already generated, call utils::browseURL(report_file) instead of rendering it again. A failure in the R Markdown setup chunk commonly reports a missing package required by the selected modules. Read that original error before using test = TRUE: recovery tests only the attached selection, preserves its order, and keeps passing modules whose parent sections also pass. Descendants of failing parents are excluded. A shorter report is an expected diagnostic outcome, not an equivalent successful full report.

Use profile = "default" or profile = "EOW" to select a known report structure explicitly. section_names() without a profile lists the complete available module pool, including alternative modules; it is not

the same as a profile's selection. Reread corrected source files with `camData(..., update = TRUE)` before rerunning `status()` or `report()`.

EOW annual REM estimates are prepared before rendering and cached in `object`. To prepare them explicitly, call `object$prepare_REM()`. Preparation uses 999 replicates and seed 42 by default; configure `object$setting$eow_rem_reps` and `object$setting$eow_rem_seed` to override these EOW-specific settings. Completed species-year fits are reused, including after `saveRDS()` and `readRDS()`. Missing fits are retried at the next preparation, with a warning identifying unavailable species-years. Changes to source tables invalidate the REM caches; `camData(..., update = TRUE)` creates a fresh object. Annual preparation is not performed during ordinary data import. General reports retain pooled REM parameters and their existing defaults.

Apart from the destination notice, rendering is quiet by default. Set `options(camtrapReport.progress = TRUE)` to show console progress for module construction, individual chunks, HTML rendering, PDF conversion, and opening the completed file. Timings include elapsed time and CPU time in the current R process; Pandoc and browser CPU time is not included in the latter. The last started chunk identifies where a long-running computation is executing. PDF page loading and printing use a 60-second timeout after browser startup; analytical computations are not interrupted by that timeout.

Value

Invisibly returns the generated report path.

See Also

`reportProfile()`, `sections()`, `section_names()`, `camData()`

Other report generation: `gui`, `camReport-method`

Examples

```
if (rmarkdown::pandoc_available()) {
# A small import example; see the vignettes for full analytical reports.
source_dataset <- system.file(
  "external", "example-small", package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
))
cm <- camData(example_dataset)
cm <- sections(cm, "introduction")
report_file <- report(
  cm,
  filename = tempfile("camtrapReport-report-"),
  view = FALSE
)
file.exists(report_file)
unlink(c(report_file, sub("\\.html$", ".Rmd", report_file)), force = TRUE)
```

```
unlink(example_dataset, recursive = TRUE, force = TRUE)
}
```

`reportProfile` *Define, import, and export report profiles*

Description

A report profile is an ordered selection of modules used to produce an ecological report and a data-status report. The ecological module pool is the collection of YAML files in `reportSections`; the data-status module pool is the separate collection in `statusSections`. These are libraries of available modules, not lists of sections that must all be rendered. A profile selects an ordered subset of either library. For example, `"report::introduction"` selects an ecological module, whereas `"status::spatial"` selects a data-status module, including when it is used in an ecological report's appendix. These prefixes select module libraries; they are not R namespace calls.

Usage

```
reportProfile(
  name,
  report = character(),
  status = character(),
  description = ""
)

read_profile(file)

write_profile(profile, file, overwrite = FALSE, object = NULL)

profile_names(object = NULL)

add_profile(object, profile, overwrite = FALSE)
```

Arguments

`name`	A non-empty profile name.
`report`	An ordered set of modules for `report()`.
`status`	An ordered set of modules for `status()`.
`description`	Optional descriptive text.
`file`	Path to a profile YAML file.
`profile`	A `reportProfile` object, a profile name registered in `object`, or a bundled profile name. `write_profile()` accepts either an object or a name.
`overwrite`	A logical value controlling whether an existing profile or file can be replaced.
`object`	A `camReport` object.

Details

Profile entries may be character vectors or data frames. Character entries use `"source::module"` notation; unqualified entries default to the report pool for `report` and the status pool for `status`. A data frame can also contain a `parent` column to override the parent declared by a module. This makes it possible, for example, to place status modules below an ecological report's appendix without modifying the shared module YAML files.

Bundled profiles are registered in each object created by `camData()`. User profiles can be registered with `add_profile()` and shared as YAML files with `read_profile()` and `write_profile()`. See the report-profile guide for a reusable custom profile and cross-library appendix example.

Value

`reportProfile()` and `read_profile()` return a `reportProfile` object. `profile_names()` returns a character vector. `add_profile()` invisibly returns the modified `camReport` object, and `write_profile()` invisibly returns the output path.

See Also

`sections()`, `section_names()`, `report()`, `status()`

Examples

```
profile <- reportProfile(
  name = "brief",
  report = c("introduction", "results"),
  status = c("abstract", "availability")
)
profile

profile_file <- tempfile(fileext = ".yml")
write_profile(profile, profile_file)
identical(read_profile(profile_file), profile)
unlink(profile_file)
```

`reportSection,character-method` *Create a report section*

Description

Create a report-section object that stores text and optional R code for a section of a `camReport` report. Report sections can be added to a report to make report generation more flexible, modular, and extensible.

Usage

```
reportSection(name, title, parent, txt, code_setting, packages, code)
```

Arguments

name	A character string giving a unique name for the report section.
title	A character string giving the section title displayed in the report. The default is "".
parent	A character string naming the parent section, or NULL (default) for a top-level section.
txt	A character vector or list containing text to include in the report section. The default is NULL.
code_setting	Optional R Markdown chunk options for the code chunk. For example, {c(echo = FALSE, results = "asis")}. The default is NULL.
packages	An optional character vector giving packages required by the code chunk. The default is NULL.
code	Optional R code supplied inside braces, for example, { plot(1:10) }.

Details

Each report section must have a unique name. If a new section with the same name as an existing section is added to a report, the existing section is replaced.

A report section can contain text only, code only, or both text and code. When the code chunk needs access to the associated camReport object, the object can be referred to as object inside the code block.

This function is useful for creating custom report modules or extending the default camtrapReport workflow with project-specific text, tables, figures, or analyses.

Only use sections from sources you trust. Text interpolation as well as R chunks can execute code during testing or report generation, with the permissions of your R process. Sections are not sandboxed; see modules.

Value

A .textSection object.

See Also

camData(), report(), updateReportSection(), testSection()

Other report sections: section_names(), testSection, .textSection-method, updateReportSection()

Examples

```
tx <- reportSection(
  name = "introduction",
  title = "Introduction",
  parent = NULL,
  txt = "This is an introduction section."
)

tx
```

```
# Create a section containing text and an R code chunk
tx2 <- reportSection(
  name = "methods",
  title = "Methods",
  parent = NULL,
  txt = "This section summarises the camera deployment design.",
  code_setting = {
    c(echo = FALSE, results = "asis", warning = FALSE)
  },
  code = {
    object$camera_setup |>
      summary()
  }
)

tx2
```

`section_names` *Select report sections*

Description

Get module names or update which sections are used by an ecological or data-status report. Profiles provide named, reusable selections without moving or duplicating the YAML module files.

Usage

```
section_names(keep, exclude, object, profile, report_type)

sections(x, n, profile, report_type)
```

Arguments

`keep`	An optional character vector of section names to keep.
`exclude`	An optional character vector of section names to exclude.
`object`	An optional `camReport` object. It is needed to inspect user-defined profiles registered in that object.
`profile`	An optional profile name or `reportProfile` object.
`report_type`	Which output selection to use: `"report"`, `"status"`, or, for `section_names()`, `"all"`.
`x`	A `camReport` object created by `camData()`.
`n`	An optional character vector of modules to include. Qualified names such as `"status::spatial"` can select from the other module pool when a profile is being edited.

Details

`section_names()` lists modules in a pool or in a profile. Set `report_type = "all"` to list both pools. Names from both pools, and names returned for a profile, use the unambiguous `"source::module"` notation.

`sections()` updates the transient module tree used for the next render. When `profile` is supplied, it can also inspect or update that registered profile. Set `report_type = "status"` to manage the data-status selection.

Value

`section_names()` and `sections(x)` return character vectors. `sections(x, n)` updates `x` and returns it invisibly. Unknown names in a partially valid `sections(x, n)` selection are ignored with a warning naming those sections. An entirely unknown selection errors.

See Also

`reportProfile()`, `report()`, `status()`, `listReportSections()`

Other report sections: `reportSection,character-method`, `testSection,.textSection-method`, `updateReportSection()`

Examples

```
section_names()
section_names(profile = "default")
section_names(report_type = "all")

# Section selection needs only a small imported dataset.
source_dataset <- system.file(
  "external", "example-small", package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
))
cm <- camData(example_dataset)
sections(cm, profile = "default")
cm <- sections(cm, c("introduction", "methods"))
unlink(example_dataset, recursive = TRUE, force = TRUE)
```

`testSection,.textSection-method`

Test a report section

Description

Render a `.textSection` object as a temporary HTML report to check whether the section text and code can be rendered successfully.

Usage

```
testSection(x, object, view)
```

Arguments

`x`	A `.textSection` object, usually created with `reportSection()`.
`object`	An optional `camReport` object created by `camData()`. The default is `NULL`.
`view`	A logical value (default `TRUE`) specifying whether the rendered temporary HTML file is opened after rendering.

Details

Security: Testing executes the section's text interpolation and R code with your R process's permissions. Test only sections from sources you trust; this is not a sandbox or a security check. See modules.

This function is mainly used to test custom report sections before they are added to a full `camReport` report. It checks whether the section text, R Markdown chunk settings, required packages, and R code can be rendered successfully.

Declared module packages are checked for availability without being attached to the R search path. Use explicit calls such as `dplyr::filter()` in custom module code instead of relying on the previously automatic attachment of declared packages. This requirement also applies when rendering full reports.

The `object` argument is only needed when the R code in the section refers to a `camReport` object.

Value

Invisibly returns the path to the rendered temporary HTML file.

See Also

`camData()`, `reportSection()`, `updateReportSection()`, `report()`

Other report sections: `reportSection,character-method`, `section_names()`, `updateReportSection()`

Examples

```
tx <- reportSection(
  name = "introduction",
  title = "Introduction",
  parent = NULL,
  txt = "This is an introduction section.",
  code = {
    plot(1:10)
  }
```

```
)

# Render the section without opening a browser
test_file <- testSection(
  tx,
  view = FALSE
)

file.exists(test_file)

# Remove the generated HTML file
unlink(test_file)
```

---

updateReportSection *Update report sections*

Description

Update the content of a report section in a camReport object, or list the report sections currently attached to the object.

Usage

```
updateReportSection(
  x,
  section,
  text,
  title,
  code,
  code_name,
  code_setting,
  packages,
  append_text,
  append_code
)

listReportSections(x)
```

Arguments

x	A camReport object created by camData().
section	A single character string giving the name or title of the report section to update.
text	An optional character value used to update the text body of the report section. The default is NULL, which leaves the text unchanged.
title	An optional character value giving a new title for the report section. The default is NULL, which leaves the title unchanged.

code	Optional R code used to update or add a code chunk. The code can be supplied inside braces.
code_name	An optional character string naming the R code chunk. The default is NULL.
code_setting	Optional R Markdown chunk options for the code chunk. The default is NULL.
packages	An optional character vector giving the R packages required by the code chunk. The default is NULL.
append_text	A logical value (default FALSE). If TRUE, the new text is appended to the existing section text; otherwise, the existing text is replaced.
append_code	A logical value (default FALSE). If TRUE, the new code is appended to the existing code chunk; otherwise, the existing code is replaced.

Details

Report sections can be identified using either their name or title. `listReportSections()` lists the sections attached to a `camReport` object and can be used to find the exact section names before updating them.

`updateReportSection()` is useful for adapting the default report content to project-specific needs, including changing section titles, replacing or appending text, and adding or updating R code chunks.

Value

`updateReportSection()` returns the updated `camReport` object invisibly. `listReportSections()` returns a data frame describing the report sections attached to the object.

See Also

`camData()`, `report()`, `reportSection()`, `info()`

Other report sections: `reportSection,character-method`, `section_names()`, `testSection,.textSection-method`

Examples

```
# Use the small dataset to demonstrate editing an existing section.
source_dataset <- system.file(
  "external",
  "example-small",
  package = "camtrapReport"
)
example_dataset <- tempfile("camtrapReport-example-")
dir.create(example_dataset)
invisible(file.copy(
  list.files(source_dataset, full.names = TRUE),
  example_dataset,
  recursive = TRUE
))

cm <- camData(example_dataset)

# List sections currently attached to the report
```

```
section_catalog <- listReportSections(cm)
head(section_catalog)

# Update the introduction
cm <- updateReportSection(
  cm,
  section = "introduction",
  title = "Project introduction",
  text = paste(
    "This report summarises camera-trap monitoring",
    "data for the example study site."
  )
)

# Confirm the updated section title
updated_catalog <- listReportSections(cm)
updated_catalog[
  updated_catalog$name == "introduction",
  c("name", "title"),
  drop = FALSE
]

unlink(example_dataset, recursive = TRUE, force = TRUE)
```

Index

∗ **camera-trap**
camData,character-method, 2
∗ **camtrapReport data**
camData,character-method, 2
∗ **optional dependencies**
install_all, 11
∗ **report generation**
gui,camReport-method, 8
report, 15
∗ **report metadata**
info,camReport-method, 9
∗ **report modules**
modules, 13
∗ **report profiles**
reportProfile, 18
∗ **report sections**
reportSection,character-method, 19
section_names, 21
testSection,.textSection-method, 22
updateReportSection, 24
∗ **spatial**
camData,character-method, 2
∗ **species**
camData,character-method, 2
.Rchunk-class *(camReport-classes)*, 6
.textSection-class *(camReport-classes)*, 6

add_Module *(modules)*, 13
add_Module,character-method *(modules)*, 13
add_profile *(reportProfile)*, 18

camData *(camData,character-method)*, 2
camData(), *6–10*, *12*, *16*, *17*, *19–21*, *23–25*
camData,character-method, 2
camInfo-class *(camReport-classes)*, 6
camR *(camReport-classes)*, 6
camReport, *2*, *4*, *8*, *9*, *14–16*, *18*, *19*, *21*, *23*, *24*
camReport *(camReport-classes)*, 6
camReport-class *(camReport-classes)*, 6
camReport-classes, 6
characterORlist-class *(camReport-classes)*, 6
characterORlistORnull-class *(camReport-classes)*, 6
characterORnull-class *(camReport-classes)*, 6

data.frameORnull-class *(camReport-classes)*, 6

empty_trash *(modules)*, 13
empty_trash,ANY-method *(modules)*, 13

gui *(gui,camReport-method)*, 8
gui(), *5*
gui,camReport-method, 8

info *(info,camReport-method)*, 9
info(), *5*, *7*, *9*, *25*
info,camReport-method, 9
info<- *(info,camReport-method)*, 9
info<-,camReport-method *(info,camReport-method)*, 9
install_all, 11
install_all,ANY-method *(install_all)*, 11

list_Modules *(modules)*, 13
list_Modules(), *16*
list_Modules,ANY-method *(modules)*, 13
listORnull-class *(camReport-classes)*, 6
listReportSections *(updateReportSection)*, 24
listReportSections(), *22*
listReportSections,camReport-method *(updateReportSection)*, 24

modules, 13, *20*, *23*
move_Module *(modules)*, 13

move_Module,character-method *(modules)*, 13

pak::pkg_install(), *12*
profile_names *(reportProfile)*, 18

read_profile *(reportProfile)*, 18
remove_Module *(modules)*, 13
remove_Module,character-method *(modules)*, 13
report, 15
report(), *5*, *7*, *9*, *10*, *12*, *15*, *18–20*, *22*, *23*, *25*
report,camReport-method *(report)*, 15
reportProfile, 18
reportProfile(), *17*, *22*
reportProfile-class *(camReport-classes)*, 6
reportSection *(reportSection,character-method)*, 19
reportSection(), *7*, *15*, *23*, *25*
reportSection,character-method, 19
restore_Module *(modules)*, 13
restore_Module,character-method *(modules)*, 13

section_names, 21
section_names(), *17*, *19*, *20*, *23*, *25*
section_names,ANY-method *(section_names)*, 21
sections *(section_names)*, 21
sections(), *17*, *19*
sections,camReport-method *(section_names)*, 21
shiny::runApp(), *8*
show,camInfo-method *(camReport-classes)*, 6
show,camReport-method *(camReport-classes)*, 6
status *(report)*, 15
status(), *4*, *5*, *7*, *9*, *10*, *12*, *15*, *18*, *19*, *22*
status,camReport-method *(report)*, 15

tempfile(), *16*
testSection *(testSection,.textSection-method)*, 22
testSection(), *15*, *20*
testSection,.textSection-method, 22

updateReportSection, 24
updateReportSection(), *10*, *15*, *20*, *22*, *23*
updateReportSection,camReport-method *(updateReportSection)*, 24

write_profile *(reportProfile)*, 18