Package: erdatools (via r-universe)

June 10, 2026

Type Package

Title Access 'ERDA' Data Archive via SFTP and HTTP

Version 0.10.0

Description Utilities for listing folders and reading image EXIF metadata from the Electronic Research Data Archive (ERDA) at Aarhus University. Supports both SFTP access via SSH ControlMaster and public anonymous HTTP URLs.

Depends R (>= 4.1.0)

License MIT + file LICENSE

Encoding UTF-8

Imports cli, fs, jsonlite, nanoparquet, purrr, tibble

Suggests testthat (>= 3.0.0), withr, pkgload, callr, mirai, httr2, httpuv, plumber, DBI, RPostgres

Config/testthat/edition 3

RoxygenNote 7.3.3

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-03 07:14:44 UTC

RemoteUrl https://gitlab.au.dk/ecos/tools/r-pkgs/erdatools

RemoteRef HEAD

RemoteSha cb0c0f13c701dbb0aa61bb572138baf0e16929b3

Contents

et_api_run *Run the erdatools REST API*

Description

Starts the plumber REST API bundled with erdatools. The API provides endpoints for ERDA directory indexing, sourceimages pipelines, and deployment management. Opens a Swagger UI in the browser by default.

Usage

```
et_api_run(port = 8000, host = "127.0.0.1", ...)
```

Arguments

port	Port to listen on.
host	Host to bind to. Use ‘"0.0.0.0"‘ to listen on all interfaces.
...	Additional arguments passed to [plumber::pr_run()].

Value

Invisibly returns the plumber router (after the server stops).

et_build_deployments *Build a deployments data frame from a CSV file*

Description

Reads a CSV file of deployment records and returns a tibble ready for [et_db_write_deployments()]. Validates required columns, normalizes values, and optionally resolves partner names to IDs.

Usage

```
et_build_deployments(csv_path, projectid, partner_map = NULL)
```

Arguments

csv_path	Character. Path to the CSV file.
projectid	Character. Project ID (e.g. ‘"ias"‘). Applied to all rows.
partner_map	Named character vector mapping partner names to IDs (e.g. ‘c("Denmark" = "5", "Sweden" = "11")‘). When supplied, the ‘partnerid‘ column is resolved and the ‘partner‘ column is dropped. When ‘NULL‘ (default) the ‘partner‘ column is kept for the caller to resolve.

Details

The ‘Partner‘ column must match values in ‘data.partners.name‘ exactly.

Required CSV columns: ‘Code‘, ‘Partner‘, ‘Location name‘, ‘Year‘, ‘Latitude‘, ‘Longitude‘.

Optional CSV columns (filled with ‘NA‘ if absent): ‘Elevation (m)‘, ‘Solar or grid power‘, ‘Trap serial number‘, ‘Comments‘.

Value

A [tibble::tibble()] with columns ‘year‘, ‘code‘, ‘locationname‘, ‘elevation‘, ‘latitude‘, ‘longitude‘, ‘power‘, ‘trapserialno‘, ‘trapstatus‘, ‘comment‘, ‘projectid‘, and either ‘partnerid‘ (when ‘partner_map‘ is supplied) or ‘partner‘ (when it is not).

Examples

```
## Not run:
# With partner resolution
partners <- DBI::dbGetQuery(con, "SELECT id, name FROM data.partners")
pmap <- setNames(partners$id, partners$name)
df <- et_build_deployments("deployments.csv", "ias", partner_map = pmap)

# Without (caller resolves later)
df <- et_build_deployments("deployments.csv", "ias")

## End(Not run)
```

---

et_build_sessions *Build a sessions data frame from grouped images*

Description

Aggregates a grouped data frame (from [et_group_motion_images()] or [et_group_snapshot_images()]) into one row per unique ‘session_key‘, matching the ‘data.sessions‘ database schema.

Usage

```
et_build_sessions(grouped, deploymentid, type = "motion")
```

Arguments

grouped	Data frame as returned by [et_group_motion_images()] or [et_group_snapshot_images()] — must contain columns ‘session_key‘ and ‘timestamp‘.
deploymentid	Character. Deployment ID (e.g. ‘"IT1#2023"‘).
type	Character. Session type: ‘"motion"‘ or ‘"snapshot"‘.

Value

Data frame with columns: ‘session_key‘ (character, retained for joining), ‘date‘ (Date), ‘deploymentid‘ (character), ‘type‘ (character), ‘starttime‘ (POSIXct), ‘endtime‘ (POSIXct). One row per unique ‘session_key‘.

Examples

```
## Not run:
grouped <- et_group_motion_images(idx)
sessions <- et_build_sessions(grouped, deploymentid = "IT1#2023", type = "motion")

## End(Not run)
```

---

et_build_sourceimages *Build a sourceimages data frame from an ERDA index*

Description

Transforms a filtered ERDA index (files only) into the ‘data.sourceimages‘ schema used by the AMI database. Constructs public URLs from the full ‘dir/name‘ path in the index and extracts timestamps from AMI or CamAlien filenames.

Usage

```
et_build_sourceimages(
  index,
  sharelink,
  http_base = "https://anon.erda.au.dk/share_redirect",
  deploymentid,
  type = "snapshot"
)
```

Arguments

index	Data frame as returned by [et_index_filter()] — must contain columns ‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘, ‘mtime‘. Should be filtered to files only (no directories).
sharelink	ERDA sharelink token.
http_base	Base URL for the ERDA share-redirect endpoint.
deploymentid	Character. Deployment ID for all rows (e.g. ‘"IT1#2023"‘).
type	Character or ‘NULL‘. Image type: ‘"snapshot"‘ or ‘"motion"‘. When ‘NULL‘, the type is derived per-image from AMI filename parsing and rows with unparseable filenames are dropped. Use ‘NULL‘ when relying on the database trigger for session grouping.

Details

The ‘index_ts‘ column records the time this function is called (i.e. the build timestamp), not the time the index was originally created.

Value

Data frame with columns: ‘index_ts‘ (character, build timestamp), ‘filename‘, ‘deploymentid‘, ‘url‘, ‘timestamp‘, ‘issnapshot‘, ‘processed‘, ‘queued‘.

Examples

```
## Not run:
idx <- et_index_filter("erda_index.parquet", extensions = c("jpg", "jpeg"))
si <- et_build_sourceimages(
  idx,
  sharelink    = Sys.getenv("ERDA_SHARELINK"),
  http_base    = "https://anon.erda.au.dk/share_redirect",
  deploymentid = "IT1#2023",
  type         = "snapshot"
)

## End(Not run)
```

---

et_build_tracks *Build a tracks data frame for database insertion*

Description

Transforms the output of [et_parse_edge_tracks()] into the ‘data.tracks‘ database schema by mapping ‘id‘ to ‘edge_id‘ and adding ‘sessionid‘, ‘algorithm‘, and ‘algorithmversion‘ columns.

Usage

```
et_build_tracks(
  parsed_tracks,
  sessionid,
  algorithm = NULL,
  algorithmversion = NULL
)
```

Arguments

parsed_tracks	Tibble as returned by [et_parse_edge_tracks()].
sessionid	Character. Session UUID to associate with all tracks.
algorithm	Character or ‘NULL‘. Algorithm name (e.g. ‘"yolov8"‘). Default ‘NULL‘.
algorithmversion	Character or ‘NULL‘. Algorithm version string. Default ‘NULL‘.

Value

A data frame with columns matching the ‘data.tracks‘ schema: ‘sessionid‘, ‘edge_id‘, ‘algorithm‘, ‘algorithmversion‘.

Examples

```
## Not run:
parsed <- et_parse_edge_tracks("results/tracks/20250508TR.csv")
tracks <- et_build_tracks(parsed, sessionid = "abc-123")

## End(Not run)
```

---

et_db_assign_sessions *Assign sessions to sourceimages*

Description

Batch UPDATE that assigns ‘sessionid‘ to sourceimages based on temporal overlap with session ‘(starttime, endtime)‘ ranges. Motion sourceimages are matched to motion sessions, and snapshot sourceimages to snapshot sessions. Only sourceimages with ‘sessionid IS NULL‘ are updated, making this function idempotent.

Usage

```
et_db_assign_sessions(dbcon, deploymentid)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]).
deploymentid	Character. Deployment ID to scope the update.

Value

The number of rows updated (invisibly).

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
et_db_assign_sessions(con, "IT1#2023")

## End(Not run)
```

`et_db_connect` *Connect to the AMI PostgreSQL database*

Description

Wrapper around [DBI::dbConnect()] using [RPostgres::Postgres()]. Connection parameters default to the environment variables used by the existing ami-indexer container (‘AMI_HOST‘, ‘AMI_USER‘, ‘AMI_PASSWORD‘, ‘AMI_DB‘).

Usage

```
et_db_connect(
  host = Sys.getenv("AMI_HOST"),
  port = 5432L,
  dbname = Sys.getenv("AMI_DB", unset = "ami_ias"),
  user = Sys.getenv("AMI_USER"),
  password = Sys.getenv("AMI_PASSWORD"),
  sslmode = "verify-full",
  sslrootcert = Sys.getenv("AMI_SSLROOTCERT")
)
```

Arguments

`host`	Database hostname. Defaults to ‘AMI_HOST‘ env var.
`port`	Database port. Defaults to ‘5432‘.
`dbname`	Database name. Defaults to ‘AMI_DB‘ env var or ‘"ami_ias"‘.
`user`	Database user. Defaults to ‘AMI_USER‘ env var.
`password`	Database password. Defaults to ‘AMI_PASSWORD‘ env var.
`sslmode`	SSL mode for the connection. Defaults to ‘"verify-full"‘ (encrypted with certificate verification).
`sslrootcert`	Path to the root CA certificate (‘.pem‘ file) used when ‘sslmode‘ is ‘"verify-full"‘ or ‘"verify-ca"‘. Defaults to the ‘AMI_SSLROOTCERT‘ env var.

Details

Requires the **DBI** and **RPostgres** packages at runtime.

Value

A DBI connection object.

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))

## End(Not run)
```

`et_db_deployment_path` *Construct the ERDA remote path for a deployment*

Description

Looks up a deployment's metadata and its associated project and partner folders to construct the canonical ERDA directory path. The path structure depends on the deployment's trap type:

Usage

```
et_db_deployment_path(dbcon, deployment_id)
```

Arguments

`dbcon`	A DBI connection object.
`deployment_id`	Character. The deployment ID (primary key).

Details

- standalone/lepisense/mothbox: /project_folder/year/partner_folder/code/ - iot: /project_folder/partner_folder/code/ (no year level; images are organised in date-based subdirectories)

Value

Character string with the remote path, e.g. ‘"/ias/2023/italy/IT1/"‘ for standalone or ‘"/storage/denmark/DK5/"‘ for IoT.

Examples

```
## Not run:
con <- et_db_connect()
path <- et_db_deployment_path(con, "IT1#2023")
# "/ias/2023/italy/IT1/"

## End(Not run)
```

`et_db_diff_deployments` *Diff deployments against the database*

Description

Queries existing ‘(code, year, projectid)‘ triples from ‘data.deployments‘ and returns only the rows in ‘df‘ that do not yet exist.

Usage

```
et_db_diff_deployments(dbcon, df)
```

Arguments

`dbcon`	A DBI connection object.
`df`	Data frame with at least ‘code‘, ‘year‘, and ‘projectid‘ columns (as returned by [et_build_deployments()]).

Value

A subset of ‘df‘ containing only new deployments. All columns are preserved.

Examples

```
## Not run:
new <- et_db_diff_deployments(con, df)
et_db_write_deployments(con, new)

## End(Not run)
```

`et_db_diff_sourceimages` *Diff sourceimages against the database*

Description

Compares a sourceimages data frame against ‘data.sourceimages‘ in the database and returns only the rows whose ‘filename‘ is not yet present. Queries are batched to stay within PostgreSQL’s parameter limit.

Usage

```
et_db_diff_sourceimages(dbcon, df, batch_size = 5000L, n_workers = 1L)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]). Used for the sequential path only; parallel workers open their own connections.
df	Data frame with at least a ‘filename‘ column (typically the output of [et_build_sourceimages()]).
batch_size	Integer. Number of filenames per query. Default ‘5000L‘.
n_workers	Integer. Number of parallel mirai workers. Default ‘1L‘. See [et_db_write_sourceimages()] for requirements.

Value

A subset of ‘df‘ containing only rows not already in the database. All original columns are preserved.

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
si  <- et_build_sourceimages(idx, sharelink, http_base = http_base, deploymentid = 1L)
new <- et_db_diff_sourceimages(con, si)
et_db_write_sourceimages(con, new)

## End(Not run)
```

---

et_db_get_deployment *Look up a deployment record from the database*

Description

Queries ‘data.deployments‘ by primary key and returns the deployment’s year, code, partner ID, and project ID.

Usage

```
et_db_get_deployment(dbcon, deployment_id)
```

Arguments

dbcon	A DBI connection object.
deployment_id	Character. The deployment ID (primary key).

Value

A named list with elements ‘year‘ (integer), ‘code‘ (character), ‘partnerid‘ (character), ‘projectid‘ (character), and ‘traptype‘ (character, e.g. ‘"standalone"‘, ‘"iot"‘, ‘"lepisense"‘, or ‘"mothbox"‘). Aborts with an error if the deployment is not found.

Examples

```
## Not run:
con <- et_db_connect()
dep <- et_db_get_deployment(con, "IT1#2023")
dep$year   # 2023
dep$code   # "IT1"

## End(Not run)
```

`et_db_get_deployment_id` *Look up a deployment ID from the database*

Description

Queries ‘data.deployments‘ for a deployment matching the given project, partner, year, and code. The deployment ID is a character string (e.g. ‘"IT1#2023"‘).

Usage

```
et_db_get_deployment_id(dbcon, projectid, partnerid, year, code)
```

Arguments

`dbcon`	A DBI connection object.
`projectid`	Character. Project identifier (e.g. ‘"ias"‘).
`partnerid`	Character. Partner ID (e.g. ‘"8"‘).
`year`	Integer. Deployment year.
`code`	Character. Deployment code (e.g. ‘"IT1"‘).

Value

Character deployment ID (e.g. ‘"IT1#2023"‘), or ‘NA_character_‘ if not found.

Examples

```
## Not run:
con <- et_db_connect()
dep_id <- et_db_get_deployment_id(con, "ias", "8", 2023L, "IT1")

## End(Not run)
```

`et_db_get_partner_folder` *Look up a partner's folder name from the database*

Description

Queries 'data.partners' for the folder associated with a partner ID.

Usage

```
et_db_get_partner_folder(dbcon, partnerid)
```

Arguments

`dbcon`	A DBI connection object.
`partnerid`	Integer. Partner ID.

Value

Character string with the partner folder, or 'NA_character_' if not found.

Examples

```
## Not run:
con <- et_db_connect()
folder <- et_db_get_partner_folder(con, 1L)

## End(Not run)
```

`et_db_get_project_folder` *Look up a project's folder name from the database*

Description

Queries 'data.projects' for the folder associated with a project ID.

Usage

```
et_db_get_project_folder(dbcon, projectid)
```

Arguments

`dbcon`	A DBI connection object.
`projectid`	Character. Project ID.

Value

Character string with the project folder, or ‘NA_character_‘ if not found.

Examples

```
## Not run:
con <- et_db_connect()
folder <- et_db_get_project_folder(con, "ias")

## End(Not run)
```

`et_db_get_sessions` *Get sessions for a deployment*

Description

Queries ‘data.sessions‘ for all sessions belonging to a deployment. Used to retrieve sessions that were created by the database trigger (migration 2034) after sourceimages were inserted.

Usage

```
et_db_get_sessions(dbcon, deploymentid)
```

Arguments

`dbcon`	A DBI connection object (from [et_db_connect()]).
`deploymentid`	Character. Deployment ID (e.g. ‘"IT1#2023"‘).

Value

Data frame with columns: ‘id‘ (character, UUID), ‘date‘ (Date), ‘type‘ (character), ‘starttime‘ (POSIXct), ‘endtime‘ (POSIXct). Zero rows if no sessions exist.

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
sessions <- et_db_get_sessions(con, "IT1#2023")

## End(Not run)
```

`et_db_list_deployments` *List deployment IDs from the database*

Description

Queries ‘data.deployments‘ for all deployments matching an optional project and/or year filter. Useful for batch operations such as running the deploy pipeline over every deployment in a given project-year.

Usage

```
et_db_list_deployments(dbcon, projectid = NULL, year = NULL)
```

Arguments

`dbcon`	A DBI connection object.
`projectid`	Character or ‘NULL‘. Project identifier (e.g. ‘"ias"‘). If ‘NULL‘, deployments from all projects are returned.
`year`	Integer or ‘NULL‘. Deployment year. If ‘NULL‘, deployments from all years are returned.

Value

Character vector of deployment IDs ordered by ‘id‘. Returns ‘character(0)‘ when no deployments match.

Examples

```
## Not run:
con <- et_db_connect()
ids <- et_db_list_deployments(con, projectid = "ias", year = 2025L)

## End(Not run)
```

`et_db_write_deployments` *Write deployments to the database*

Description

Performs batch INSERT into ‘data.deployments‘. Each deployment receives a UUID generated by ‘uuid_generate_v4()‘ in PostgreSQL. Geographic coordinates are converted to a PostGIS point via ‘ST_SetSRID(ST_MakePoint(lon, lat), 4326)‘.

Usage

```
et_db_write_deployments(dbcon, df, batch_size = 250L)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]).
df	Data frame with columns: ‘year‘, ‘code‘, ‘locationname‘, ‘elevation‘, ‘latitude‘, ‘longitude‘, ‘trapserialno‘, ‘trapstatus‘, ‘traptype‘, ‘comment‘, ‘power‘, ‘partnerid‘, ‘projectid‘.
batch_size	Integer. Rows per INSERT statement (default 250).

Details

Use [et_db_diff_deployments()] beforehand to avoid inserting duplicates.

Value

Total number of rows inserted (invisibly).

Examples

```
## Not run:
df <- et_build_deployments("deployments.csv", "ias", partner_map = pmap)
new <- et_db_diff_deployments(con, df)
et_db_write_deployments(con, new)

## End(Not run)
```

---

et_db_write_sessions *Write sessions to the database*

Description

Performs batch INSERT into ‘data.sessions‘ with ON CONFLICT upsert on ‘(date, deploymentid, type)‘. Returns the input data frame augmented with ‘id‘ (UUID) from the database.

Usage

```
et_db_write_sessions(dbcon, df, batch_size = 250L, n_workers = 1L)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]). Used for the sequential path and the final ID-fetch; parallel workers open their own connections.
df	Data frame with columns: ‘date‘, ‘deploymentid‘, ‘type‘, ‘starttime‘, ‘endtime‘. Typically the output of [et_build_sessions()].
batch_size	Integer. Number of rows per INSERT statement. Default ‘250L‘.
n_workers	Integer. Number of parallel mirai workers. Default ‘1L‘. See [et_db_write_sourceimages()] for requirements.

Value

The input data frame with an ‘id‘ column (character UUID) populated from the database.

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
sessions <- et_build_sessions(grouped, "IT1#2023", type = "motion")
sessions_with_ids <- et_db_write_sessions(con, sessions)

## End(Not run)
```

---

et_db_write_sourceimages

Write sourceimages data frame to the database

Description

Performs batch INSERT into ‘data.sourceimages‘ with ON CONFLICT upsert on ‘(filename, deploymentid)‘. Existing rows are updated with the new values. Uses parameterized queries via [DBI::dbExecute()].

Usage

```
et_db_write_sourceimages(dbcon, df, batch_size = 250L, n_workers = 1L)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]). Used for the sequential path only; parallel workers open their own connections via env vars.
df	Data frame with columns: ‘filename‘, ‘deploymentid‘, ‘url‘, ‘timestamp‘, ‘issnapshot‘, ‘processed‘, ‘queued‘.
batch_size	Integer. Number of rows per INSERT statement. Default ‘250L‘.

n_workers	Integer. Number of parallel mirai workers. Default ‘1L‘ (sequential). When ‘> 1‘, batches are dispatched via [mirai::mirai_map()] and each worker opens and closes its own DB connection. Requires the **mirai** package and an installed (not just loaded) **erdatools**.

Value

The total number of rows affected (invisibly).

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
et_db_write_sourceimages(con, sourceimages_df)
et_db_write_sourceimages(con, sourceimages_df, n_workers = 4L)

## End(Not run)
```

---

et_db_write_tracks *Write tracks to the database*

Description

Performs batch INSERT into ‘data.tracks‘ with ON CONFLICT upsert on ‘(sessionid, edge_id)‘. Typically used to write edge-processed tracking data from IoT AMI traps.

Usage

```
et_db_write_tracks(dbcon, df, batch_size = 250L)
```

Arguments

dbcon	A DBI connection object (from [et_db_connect()]).
df	Data frame with columns: ‘sessionid‘, ‘edge_id‘, ‘algorithm‘, ‘algorithmversion‘. Typically the output of [et_build_tracks()].
batch_size	Integer. Number of rows per INSERT statement. Default ‘250L‘.

Value

Total number of rows affected (invisibly).

Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
parsed <- et_parse_edge_tracks("results/tracks/20250508TR.csv")
tracks <- et_build_tracks(parsed, sessionid = "abc-123")
et_db_write_tracks(con, tracks)

## End(Not run)
```

---

et_exif_tags *Resolve the EXIF tags to extract*

Description

Returns the vector of ExifTool tag names passed to [exifr::read_exif()]. When the environment variable ‘EXIF_TAGS‘ is set (comma-separated tag names), it is used instead of the built-in defaults. ‘"FileName"‘ is always prepended so output parquet files retain their original name.

Usage

```
et_exif_tags()
```

Details

If ‘EXIF_TAGS‘ is set but produces no valid tag names after splitting and trimming (e.g. ‘EXIF_TAGS=" , , "‘), a warning is emitted and the built-in defaults are used.

Default tags (when ‘EXIF_TAGS‘ is unset): ‘"FileName"‘, ‘"Model"‘, ‘"CameraSerialNumber"‘, ‘"XResolution"‘, ‘"YResolution"‘, ‘"ImageWidth"‘, ‘"ImageHeight"‘, ‘"UserComment"‘, ‘"Duration"‘, ‘"VideoFrameRate"‘.

Value

Character vector of tag names, always starting with ‘"FileName"‘.

---

et_filter_media_files *Filter a character vector to supported media file extensions*

Description

Keeps filenames ending in ‘.jpg‘, ‘.jpeg‘, ‘.tif‘, ‘.tiff‘, ‘.mp4‘, ‘.mov‘, or ‘.avi‘ (all case-insensitive).

Usage

```
et_filter_media_files(filenames)
```

Arguments

filenames	Character vector of file paths or names.

Value

Subset of ‘filenames‘ matching a supported extension.

---

et_filter_stray_images

Filter out images with timestamps outside the deployment year range

---

Description

Removes rows from a grouped image data frame where the ‘timestamp‘ year falls outside ‘[year, year + 1]‘. Deployments can span a year boundary (e.g. a 2025 deployment may have images from early 2026), so ‘year + 1‘ is included.

Usage

```
et_filter_stray_images(grouped, year, log_path = NULL)
```

Arguments

grouped	Data frame as returned by [et_group_motion_images()] or [et_group_snapshot_images()]. Must contain columns ‘dir‘, ‘name‘, and ‘timestamp‘ (POSIXct).
year	Integer. The deployment year.
log_path	Character or ‘NULL‘. If non-‘NULL‘, full ERDA paths of filtered images are written to this file (one per line). Parent directories are created if needed.

Details

Optionally writes the ERDA paths of removed images to a log file.

Value

A named list:

kept Data frame of rows whose timestamps are within range.

removed Data frame of rows that were filtered out.

Examples

```
## Not run:
grouped <- et_group_motion_images(idx)
result <- et_filter_stray_images(grouped, year = 2025)
grouped <- result$kept

## End(Not run)
```

`et_filter_stray_sourceimages` *Filter out sourceimages with timestamps outside the deployment year range*

Description

Removes rows from a sourceimages data frame where the ‘timestamp‘ year falls outside ‘[year, year + 1]‘. This is the sourceimages-schema equivalent of [et_filter_stray_images()], operating on character timestamps and the ‘filename‘ column instead of grouped data frames.

Usage

```
et_filter_stray_sourceimages(sourceimages, year, log_path = NULL)
```

Arguments

`sourceimages`	Data frame as returned by [et_build_sourceimages()]. Must contain columns ‘filename‘ and ‘timestamp‘ (character, format ‘"YYYY-MM-DD HH:MM:SS"‘).
`year`	Integer. The deployment year.
`log_path`	Character or ‘NULL‘. If non-‘NULL‘, filenames of removed images are written to this file (one per line). Parent directories are created if needed.

Value

A named list:

kept Data frame of rows whose timestamps are within range.

removed Data frame of rows that were filtered out.

Examples

```
## Not run:
si <- et_build_sourceimages(idx, sharelink = "TOKEN",
  deploymentid = "IT1#2023", type = NULL)
result <- et_filter_stray_sourceimages(si, year = 2023)
si <- result$kept

## End(Not run)
```

`et_group_motion_images` *Group motion images into noon-to-noon sessions*

Description

Takes an ERDA index data frame (as returned by [et_index_filter()]) and groups motion images into **sessions** using noon-to-noon grouping.

Usage

```
et_group_motion_images(index)
```

Arguments

`index`	Data frame with columns ‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘, ‘mtime‘ (as returned by [et_index_filter()]).

Details

A session spans from noon UTC on one day to noon UTC on the next. The ‘session_key‘ column is the calendar date of the noon boundary, so an image at 22:00 on Aug 31 and another at 06:00 on Sep 1 both receive ‘session_key = "2023_08_31"‘. An image at 13:00 on Sep 1 starts a new session with key ‘"2023_09_01"‘.

Only motion images (as identified by [et_parse_ami_filename()]) are retained.

Value

The filtered data frame with three added columns:

session_key Character. Session date as ‘"YYYY_MM_DD"‘.

timestamp POSIXct (UTC). Parsed from the AMI filename.

trigger Character. Trigger counter from the filename.

Examples

```
## Not run:
idx <- et_index_filter("motion_index.parquet", extensions = c("jpg", "jpeg"))
grouped <- et_group_motion_images(idx)
table(grouped$session_key)

## End(Not run)
```

`et_group_snapshot_images` *Group snapshot images into noon-to-noon sessions*

Description

Takes an ERDA index data frame (as returned by [et_index_filter()]) and groups snapshot images into **sessions** using noon-to-noon grouping.

Usage

```
et_group_snapshot_images(index)
```

Arguments

`index`	Data frame with columns ‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘, ‘mtime‘ (as returned by [et_index_filter()]).

Details

A session spans from noon UTC on one day to noon UTC on the next. The ‘session_key‘ column is the calendar date of the noon boundary, so an image at 22:00 on Aug 31 and another at 06:00 on Sep 1 both receive ‘session_key = "2023_08_31"‘. An image at 13:00 on Sep 1 starts a new session with key ‘"2023_09_01"‘.

Only snapshot images (as identified by [et_parse_ami_filename()]) are retained.

Value

The filtered data frame with two added columns:

session_key Character. Session date as ‘"YYYY_MM_DD"‘.

timestamp POSIXct (UTC). Parsed from the AMI filename.

Examples

```
## Not run:
idx <- et_index_filter("snapshot_index.parquet", extensions = c("jpg", "jpeg"))
grouped <- et_group_snapshot_images(idx)
table(grouped$session_key)

## End(Not run)
```

et_img_url *Construct a public anonymous URL for an ERDA image*

Description

Builds the HTTPS URL used to access a single image file through ERDA's anonymous share-redirect endpoint. This is a pure string operation and requires no network connection. Slashes between path components are normalised automatically.

Usage

```
et_img_url(
  sharelink,
  remote_dir,
  country,
  folder,
  filename,
  http_base = "https://anon.erda.au.dk/share_redirect"
)
```

Arguments

sharelink	ERDA sharelink token.
remote_dir	Remote root directory, i.e. the path relative to the sharelink root (e.g. ‘"/storage/onestop/2025/"‘). Leading and trailing slashes are stripped before joining.
country	Country subdirectory name (e.g. ‘"portugal"‘).
folder	Dated subfolder name (e.g. ‘"2025_07_10-11_10"‘).
filename	Image file name (e.g. ‘"img.jpg"‘).
http_base	Base URL of the ERDA share-redirect endpoint. Defaults to ‘"https://anon.erda.au.dk/share_redirect"‘.

Value

A length-1 character vector containing the full URL.

Examples

```
et_img_url(
  sharelink  = "TOKEN",
  remote_dir = "/storage/2025/",
  country    = "portugal",
  folder     = "2025_07_10-11_10",
  filename   = "img.jpg"
)
```

`et_index_dir` *Build a file index of a remote ERDA directory tree*

Description

Traverses the directory tree rooted at ‘remote_path‘ using BFS, calling ‘ls -la‘ on each directory via [et_sftp_batch()]. Results are stored in a parquet file at ‘index_path‘ with columns ‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘ (integer, file size in bytes; ‘NA‘ for directories), and ‘mtime‘ (character, last-modified timestamp from ‘ls -la‘, e.g. ‘"Jul 10 11:10"‘ or ‘"Jan 5 2024"‘).

Usage

```
et_index_dir(
  conn,
  remote_path,
  index_path,
  checkpoint_every = 25L,
  n_workers = 1L
)
```

Arguments

`conn`	Connection object returned by [et_sftp_connect()].
`remote_path`	Remote root path to index.
`index_path`	Local path for the output parquet file.
`checkpoint_every`	Integer. Rewrite the parquet every this many directories. Default ‘25L‘.
`n_workers`	Integer. Number of parallel mirai workers for wave-based BFS. Default ‘1L‘ (sequential). When ‘> 1‘, each wave of directories is listed concurrently – all workers share the parent ControlMaster socket. Requires the **mirai** package and an installed **erdatools**.

Details

Resume: if ‘index_path‘ already exists, the index is loaded and any directory already present in ‘unique(index$dir)‘ is skipped. Previously indexed directories are used to reconstruct the BFS queue without additional SFTP calls, so resuming after an interrupted run is fast.

Checkpointing: the parquet is rewritten every ‘checkpoint_every‘ directories. At most ‘checkpoint_every‘ directories of work can be lost if the connection drops between checkpoints.

Value

The full index data frame (invisibly).

Examples

```
## Not run:
conn <- et_sftp_connect(Sys.getenv("ERDA_SHARELINK"))
on.exit(conn$disconnect(), add = TRUE, after = FALSE)
idx <- et_index_dir(conn, "/storage/2025", "./erda_index.parquet")
idx <- et_index_dir(conn, "/storage/2025", "./erda_index.parquet", n_workers = 4L)

## End(Not run)
```

`et_index_filter` *Filter an ERDA index to files matching given extensions*

Description

Removes directory rows from the index and optionally filters by file extension. Returns a filtered data frame with the same schema as the input.

Usage

```
et_index_filter(index, extensions = NULL)
```

Arguments

`index`	A data frame as returned by [et_index_dir()], or a character string of length 1 giving the path to a parquet file.
`extensions`	Character vector of file extensions to keep, e.g. ‘c("jpg", "jpeg")‘ or ‘c(".tif")‘. A leading dot is added automatically if missing. Matching is case-insensitive. ‘NULL‘ (default) keeps all files.

Value

Data frame with the same columns as the input index (‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘, ‘mtime‘), filtered to files only.

Examples

```
## Not run:
# From a parquet file
filtered <- et_index_filter("erda_index.parquet", extensions = c("jpg", "jpeg"))

# From an in-memory data frame
idx <- et_index_dir(conn, "/storage/2025", "erda_index.parquet")
filtered <- et_index_filter(idx, extensions = "jpg")

# All files (no extension filter)
all_files <- et_index_filter("erda_index.parquet")

## End(Not run)
```

et_index_update *Detect additions and removals in an existing ERDA index*

Description

Re-runs ‘ls -la‘ on every directory in the existing index, diffs the listings against the stored entries, and returns a named list describing the changes. Newly discovered subdirectories are recursively indexed via [et_index_dir()].

Usage

```
et_index_update(conn, remote_path, index_path, update = TRUE)
```

Arguments

conn	Connection object returned by [et_sftp_connect()].
remote_path	Remote root path (used when recursively indexing new subdirectories).
index_path	Local path to the existing parquet index file.
update	Logical. If ‘TRUE‘ (default), the index is updated in place with the detected changes.

Value

Named list ‘list(added = <data.frame>, removed = <data.frame>)‘ (invisibly). Each data frame has columns ‘dir‘, ‘name‘, ‘is_dir‘, ‘size‘ (integer, bytes), ‘mtime‘ (character timestamp).

Examples

```
## Not run:
conn <- et_sftp_connect(Sys.getenv("ERDA_SHARELINK"))
on.exit(conn$disconnect(), add = TRUE, after = FALSE)
delta <- et_index_update(conn, "/storage/2025", "./erda_index.parquet")

## End(Not run)
```

et_job_create *Create a new job record*

Description

Creates a job in the in-memory job store with status ‘"queued"‘.

Usage

```
et_job_create(type, params = list())
```

Arguments

type	Character. Job type (e.g. ‘"index_full"‘, ‘"index_update"‘, ‘"sourceimages_write"‘).
params	Named list of job parameters.

Value

Character string: the job ID.

Examples

```
id <- et_job_create("index_full", list(remote_path = "/storage/2025"))
et_job_get(id)
```

---

et_job_get *Get a job by ID*

Description

If the job has a background process handle, its status is synced from the process state before returning.

Usage

```
et_job_get(id)
```

Arguments

id	Character. Job ID.

Value

Named list with job details, or ‘NULL‘ if not found.

Examples

```
id <- et_job_create("test", list())
et_job_get(id)
```

et_job_list *List all jobs*

Description

Syncs status from background process handles before returning.

Usage

```
et_job_list()
```

Value

A data frame with columns: ‘id‘, ‘type‘, ‘status‘, ‘created_at‘, ‘updated_at‘. Returns an empty data frame if no jobs exist.

Examples

```
et_job_list()
```

et_job_set_process *Attach a background mirai handle to a job*

Description

Stores a [mirai::mirai()] handle on the job and sets the status to ‘"running"‘. The handle is polled by [et_job_get()] and [et_job_list()] to derive the current status.

Usage

```
et_job_set_process(id, process)
```

Arguments

id	Character. Job ID.
process	A ‘mirai‘ object (from [mirai::mirai()]).

Value

The updated job list (invisibly).

et_job_update *Update a job's status*

Description

Update a job's status

Usage

```
et_job_update(id, status, result = NULL, error = NULL)
```

Arguments

id	Character. Job ID.
status	Character. New status: ‘"queued"‘, ‘"running"‘, ‘"completed"‘, or ‘"failed"‘.
result	Optional result object to attach.
error	Optional error message to attach.

Value

The updated job list (invisibly).

Examples

```
id <- et_job_create("test", list())
et_job_update(id, "running")
et_job_update(id, "completed", result = list(rows = 100))
```

et_parquet_folders_done *Check which folders already have a parquet file*

Description

Given a character vector of folder names and a local parquet directory, returns a named logical vector indicating which folders already have a corresponding ‘.parquet‘ file. This is a pure function: it performs no network I/O.

Usage

```
et_parquet_folders_done(folders, parquet_dir)
```

Arguments

folders	Character vector of folder names. If full paths are supplied, [base::basename()] is applied before looking up the parquet file.
parquet_dir	Path to the local parquet directory.

Value

Named logical vector (names equal to ‘folders‘). ‘TRUE‘ means a ‘.parquet‘ file already exists for that folder.

Examples

```
tmp <- tempdir()
writeLines("", fs::path(tmp, "2025_07_10-11_10.parquet"))
et_parquet_folders_done(c("2025_07_10-11_10", "2025_07_11-08_00"), tmp)
```

---

et_parse_ami_filename *Parse AMI (IAS) image filenames*

Description

AMI traps produce five filename formats:

Usage

```
et_parse_ami_filename(filenames)
```

Arguments

filenames	Character vector of filenames (with or without extension).

Details

- **Motion (standalone)**: ‘<YYYYMMDDHHmmss>-<sequence>-<trigger>.ext‘ (e.g. ‘20230831124747-00-01.jpg‘) - **Motion (IoT)**: ‘<YYYYMMDDHHmmss>.ext‘ (e.g. ‘20250508104842.jpg‘) — bare 14-digit timestamp, no sequence or trigger counter - **Snapshot (with camera id)**: ‘<camera_id>-<YYYYMMDDHHmmss>-<type>.ext‘ (e.g. ‘01-20230831225959-snapshot.jpg‘) - **Snapshot (no camera id)**: ‘<YYYYMMDDHHmmss>-<type>.ext‘ (e.g. ‘20250527110000-snapshot.jpg‘) - **Mothbox**: ‘<trap_name>_<YYYY>_<MM>_<DD>__<HH>_<MM>_<SS>_<HDR>.ext‘ (e.g. ‘wryNinox_2026_05_29__23_02_29_HDR0.jpg‘) — scheduled captures with underscore-delimited date/time separated by double underscore

This function extracts the trigger counter (or camera ID for snapshots), timestamp, and image type from each filename. All formats are detected automatically. Filenames that match none of the patterns produce ‘NA‘ values for all columns.

Value

A data frame with columns ‘trigger‘ (character: trigger counter for motion images, camera ID for snapshot images that include one, or ‘NA‘), ‘timestamp‘ (character, ‘YYYY-MM-DD HH:MM:SS‘), ‘type‘ (character, typically ‘"snapshot"‘ or ‘"motion"‘), and ‘sequence‘ (character, motion sequence number or ‘NA‘ for snapshots). One row per filename.

Examples

```
et_parse_ami_filename(c(
  "01-20230831225959-snapshot.jpg",
  "20250527110000-snapshot.jpg",
  "20230831124747-00-01.jpg",
  "20250508104842.jpg",
  "wryNinox_2026_05_29__23_02_29_HDR0.jpg"
))
```

`et_parse_camalien_filename` *Parse key-value pairs encoded in a CamAlien image filename*

Description

CamAlien filenames encode metadata as underscore-delimited key-value pairs (e.g. ‘CT_1234_GT_5678_lon_12.34_lat_56.
This function strips the file extension and splits on ‘_‘, treating odd-positioned tokens as keys and even-positioned tokens as values.

Usage

```
et_parse_camalien_filename(filenames)
```

Arguments

`filenames`	Character vector of filenames (with or without extension).

Details

If the number of tokens is odd, the last value is padded with ‘NA_character_‘ so that key-value pairing still works.

Known numeric columns (‘alt‘, ‘vel‘, ‘expt‘, ‘gain‘) are coerced to ‘double‘. Coordinates (‘lon‘, ‘lat‘) and timestamps (‘CT‘, ‘GT‘) are kept as ‘character‘ to preserve their original string representation. All other columns are ‘character‘.

Note: Values containing underscores will cause incorrect key-value pairing, since ‘_‘ is also the key-value and pair delimiter.

Value

A [tibble::tibble()] with one row per filename and one column per key found in the filenames. Columns not present in a given filename are filled with 'NA'. Column types: 'alt', 'vel', 'expt', 'gain' are 'double'; all others are 'character'.

Examples

```
et_parse_camalien_filename(c(
  "CT_1234_GT_5678_lon_12.34_lat_56.78_alt_120.5_vel_0.0_expt_0.01_gain_1.4.jpg",
  "CT_9999_GT_0000_lon_8.10_lat_55.40_alt_80.0_vel_2.3_expt_0.02_gain_2.0.jpg"
))
```

---

et_parse_edge_detections

Parse edge detection CSV

Description

Reads a detection CSV produced by IoT AMI trap edge processing (e.g. 'results/detect/YYYYMMDD.csv'). Parses timestamps and coerces column types.

Usage

```
et_parse_edge_detections(path)
```

Arguments

`path`	Character. Path to the detection CSV file.

Value

A [tibble::tibble()] with columns:

year Integer. Capture year.

trap Character. Trap identifier.

timestamp POSIXct (UTC). Capture time.

detect_conf Double. Object detection confidence.

detect_id Integer. Detection ID within image.

x1, y1, x2, y2 Double. Bounding box coordinates.

filename Character. Source image filename.

order_label Character. Taxonomic order label.

order_id Integer. Order ID.

order_conf Double. Order classification confidence.

above_th Logical. Whether confidence exceeds threshold.

key Integer. Detection key (links to track details).

species_label Character. Species label.

species_id Integer. Species ID.

species_conf Double. Species classification confidence.

Examples

```
## Not run:
detections <- et_parse_edge_detections("results/detect/20250508.csv")

## End(Not run)
```

`et_parse_edge_tracks` *Parse edge track summary CSV*

Description

Reads a track summary CSV produced by IoT AMI trap edge processing (e.g. 'results/tracks/YYYYMMDDTR.csv'). Parses timestamps and coerces column types.

Usage

```
et_parse_edge_tracks(path)
```

Arguments

`path`	Character. Path to the track summary CSV file.

Value

A [tibble::tibble()] with columns:

id Integer. Track ID from edge processor.

starttime POSIXct (UTC). Start of track.

endtime POSIXct (UTC). End of track.

duration Integer. Duration in seconds.

class Character. Species label (majority vote).

counts Integer. Number of frames in track.

percentage Double. Classification consistency in percent.

size Double. Mean bounding box area in pixels squared.

distance Double. Total pixel displacement.

Examples

```
## Not run:
tracks <- et_parse_edge_tracks("results/tracks/20250508TR.csv")

## End(Not run)
```

`et_process_edge_tracks` *Download, parse, and write edge tracks to the database*

Description

Finds track summary CSVs in an ERDA index, downloads each via SFTP, parses them with [et_parse_edge_tracks()], maps tracks to motion sessions using the noon-to-noon date boundary, and writes to the database via [et_db_write_tracks()].

Usage

```
et_process_edge_tracks(conn, dbcon, idx_raw, sessions)
```

Arguments

`conn`	SFTP connection object from [et_sftp_connect()].
`dbcon`	A DBI connection object.
`idx_raw`	Data frame. Raw ERDA index (as returned by [et_index_dir()]).
`sessions`	Data frame. Sessions with ‘id‘, ‘date‘, and ‘type‘ columns (as returned by [et_db_write_sessions()]).

Value

Integer. Total number of track rows written.

`et_sftp_batch` *Run an SFTP batch session*

Description

Executes one SFTP batch session over the existing ControlMaster connection returned by [et_sftp_connect()], returning standard output as a character vector.

Usage

```
et_sftp_batch(conn, commands, stdout = TRUE, stderr = FALSE)
```

Arguments

conn	Connection object returned by [et_sftp_connect()].
commands	Character vector of SFTP commands, passed to the session via ‘stdin‘.
stdout	Passed to [base::system2()]. ‘TRUE‘ (default) captures output as a character vector.
stderr	Passed to [base::system2()]. ‘FALSE‘ (default) discards stderr.

Value

Character vector of stdout lines (when ‘stdout = TRUE‘), or the integer exit code (when ‘stdout = FALSE‘).

Examples

```
## Not run:
conn <- et_sftp_connect(Sys.getenv("ERDA_SHARELINK"))
on.exit(conn$disconnect(), add = TRUE, after = FALSE)
lines <- et_sftp_batch(conn, paste0("ls -1 /storage/2025/"))

## End(Not run)
```

---

et_sftp_connect *Connect to ERDA via SSH ControlMaster*

Description

Establishes a persistent SSH connection to ERDA using an SSH ControlMaster socket. Authentication uses ‘SSH_ASKPASS‘ — no ‘sshpass‘ is required. The function waits up to 5 seconds for the socket to appear and calls [cli::cli_abort()] on timeout.

Usage

```
et_sftp_connect(
  sharelink,
  host = "io.erda.au.dk",
  port = "2222",
  tmp_root = fs::path(tempdir(), "sftp_imgs")
)
```

Arguments

sharelink	ERDA sharelink token, used as both the SSH user name and password.
host	ERDA SFTP hostname. Defaults to ‘"io.erda.au.dk"‘.
port	ERDA SFTP port. Defaults to ‘"2222"‘.
tmp_root	Local directory used for the SSH control socket and temporary downloads. Created if it does not exist. Defaults to a subdirectory of [base::tempdir()].

Details

The returned connection object contains a ‘$disconnect()‘ closure that kills any tracked child SFTP processes, closes the ControlMaster, and removes the temporary directory. Register it in the calling frame with:

“‘r conn <- et_sftp_connect(sharelink, host, port) on.exit(conn$disconnect(), add = TRUE, after = FALSE) “‘

Value

A named list (connection object) with elements:

‘ctl_opts‘ Character vector of SSH ‘-o‘ flags for multiplexing.

‘remote_target‘ String ‘"<sharelink>@<host>"‘.

‘tmp_root‘ Path to the temporary directory.

‘port‘ The SFTP port string.

‘add_child_pid‘ ‘function(pid)‘ — register a background SFTP PID for cleanup on disconnect.

‘remove_child_pids‘ ‘function(pids)‘ — deregister finished PIDs.

‘disconnect‘ Parameterless function — kill tracked children, close the ControlMaster, and remove ‘tmp_root‘.

Examples

```
## Not run:
conn <- et_sftp_connect(
  sharelink = Sys.getenv("ERDA_SHARELINK"),
  host      = Sys.getenv("ERDA_SFTP_HOST", unset = "io.erda.au.dk"),
  port      = Sys.getenv("ERDA_SFTP_PORT", unset = "2222")
)
on.exit(conn$disconnect(), add = TRUE, after = FALSE)

## End(Not run)
```

---

et_sftp_list_dated_subfolders

List dated subfolders within a country path

Description

Calls [et_sftp_list_folders()] and filters results to entries whose [base::basename()] matches the ‘YYYY_MM_DD-HH_MM‘ pattern used by CamAlien onboard systems.

Usage

```
et_sftp_list_dated_subfolders(conn, country_path)
```

Arguments

conn	Connection object returned by [et_sftp_connect()].
country_path	Remote SFTP path to the country folder.

Value

Character vector of matching folder names (or full paths, depending on what the SFTP server returns).

Examples

```
## Not run:
conn <- et_sftp_connect(Sys.getenv("ERDA_SHARELINK"))
on.exit(conn$disconnect(), add = TRUE, after = FALSE)
subfolders <- et_sftp_list_dated_subfolders(conn, "/storage/2025/portugal")

## End(Not run)
```

---

et_sftp_list_folders *List entries at a remote SFTP path*

Description

Runs 'ls -1 <remote_path>' over the SFTP connection and returns the non-blank, non-prompt lines.

Usage

```
et_sftp_list_folders(conn, remote_path)
```

Arguments

conn	Connection object returned by [et_sftp_connect()].
remote_path	Remote SFTP path to list.

Value

Character vector of folder/file names returned by the server.

Examples

```
## Not run:
conn <- et_sftp_connect(Sys.getenv("ERDA_SHARELINK"))
on.exit(conn$disconnect(), add = TRUE, after = FALSE)
folders <- et_sftp_list_folders(conn, "/storage/2025/portugal")

## End(Not run)
```

parse_detections *Parse UserComment JSON into a list of per-image detection data frames*

Description

Each element of ‘user_comment‘ is expected to be a JSON string of the form: “‘json "gain": "1.4", "detections": ["1359658 Morus alba L. 9.97668 8.829123", ...] “‘ Each detection string has the layout: ‘<plantnet_id> <species tokens...> <probability> <score>‘

Usage

```
parse_detections(user_comment)
```

Arguments

user_comment	Character vector of UserComment EXIF values (one per image row).

Details

The function is designed to be used with [tidyr::unnest()] on a list column. NA, empty, or malformed entries yield a zero-row data frame.

Value

A list of [tibble::tibble()]s, one per element of ‘user_comment‘, each with columns ‘plantnet_id‘ (integer), ‘species‘ (character), ‘probability‘ (double), ‘score‘ (double).

parse_sftp_ls_la *Parse raw SFTP ls -la output*

Description

Strips ‘sftp>‘ prompt lines, blank lines, and ‘.‘/‘..‘ entries from the raw character vector returned by [et_sftp_batch()] when running ‘ls -la <path>‘. Parses the permission string to determine entry type.

Usage

```
parse_sftp_ls_la(raw_lines)
```

Arguments

raw_lines	Character vector of raw SFTP stdout lines.

Details

Each line must have at least 9 whitespace-separated fields: ‘<perms> <links> <owner> <group> <size> <month> <day> <time> <name...>‘. Lines with fewer fields are silently dropped.

For symlinks (‘lrwxrwxrwx‘), the ‘ -> target‘ suffix is stripped from the name and the entry is treated as a regular file (‘is_dir = FALSE‘).

Value

Data frame with columns ‘name‘ (character), ‘is_dir‘ (logical), ‘size‘ (integer – file size in bytes, ‘NA‘ for directories), ‘mtime‘ (character – last-modified timestamp from ‘ls -la‘, fields 6-8 joined with a space, e.g. ‘"Jul 10 11:10"‘ or ‘"Jan 5 2024"‘).

Index