

# Package: erdatools (via r-universe)

October 7, 2026

**Type** Package

**Title** Access 'ERDA' Data Archive via SFTP and HTTP

**Version** 0.14.1

**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 (>= 4.6.0)

**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, uuid

**Config/testthat/edition** 3

**Config/roxygen2/markdown** TRUE

**Config/roxygen2/version** 8.0.0

**Config/pak/sysreqs** cmake make libuv1-dev

**Repository** <https://ldalby.r-universe.dev>

**Date/Publication** 2026-10-06 09:56:32 UTC

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

**RemoteRef** HEAD

**RemoteSha** 1fc70072725d2e86b35e40ffa077c46751fbfd95

## Contents

|                                     |   |
|-------------------------------------|---|
| et_api_run . . . . .                | 3 |
| et_build_camalien_images . . . . .  | 4 |
| et_build_classifications . . . . .  | 5 |
| et_build_deployments . . . . .      | 6 |
| et_build_detection_tracks . . . . . | 7 |

|                                        |    |
|----------------------------------------|----|
| et_build_detections . . . . .          | 8  |
| et_build_sessions . . . . .            | 9  |
| et_build_sourceimages . . . . .        | 10 |
| et_build_tracks . . . . .              | 11 |
| et_camalien_db_connect . . . . .       | 12 |
| et_camalien_db_diff_images . . . . .   | 13 |
| et_camalien_db_get_partner . . . . .   | 14 |
| et_camalien_db_get_session . . . . .   | 15 |
| et_camalien_db_write_images . . . . .  | 15 |
| et_check_stray_deployments . . . . .   | 16 |
| et_db_assign_sessions . . . . .        | 17 |
| et_db_connect . . . . .                | 18 |
| et_db_deployment_path . . . . .        | 19 |
| et_db_diff_deployments . . . . .       | 20 |
| et_db_diff_sourceimages . . . . .      | 21 |
| et_db_get_deployment . . . . .         | 22 |
| et_db_get_deployment_id . . . . .      | 22 |
| et_db_get_partner_folder . . . . .     | 23 |
| et_db_get_project_folder . . . . .     | 24 |
| et_db_get_sessions . . . . .           | 24 |
| et_db_get_sourceimages . . . . .       | 25 |
| et_db_get_tracks . . . . .             | 26 |
| et_db_list_deployments . . . . .       | 26 |
| et_db_write_classifications . . . . .  | 27 |
| et_db_write_deployments . . . . .      | 28 |
| et_db_write_detection_tracks . . . . . | 29 |
| et_db_write_detections . . . . .       | 30 |
| et_db_write_sessions . . . . .         | 31 |
| et_db_write_sourceimages . . . . .     | 31 |
| et_db_write_tracking_tasks . . . . .   | 32 |
| et_db_write_tracks . . . . .           | 33 |
| et_exif_tags . . . . .                 | 34 |
| et_filter_media_files . . . . .        | 35 |
| et_filter_stray_images . . . . .       | 35 |
| et_filter_stray_sourceimages . . . . . | 36 |
| et_group_motion_images . . . . .       | 37 |
| et_group_snapshot_images . . . . .     | 38 |
| et_img_url . . . . .                   | 39 |
| et_index_dir . . . . .                 | 40 |
| et_index_filter . . . . .              | 41 |
| et_index_update . . . . .              | 42 |
| et_job_create . . . . .                | 43 |
| et_job_get . . . . .                   | 43 |
| et_job_list . . . . .                  | 44 |
| et_job_progress_file . . . . .         | 44 |
| et_job_set_process . . . . .           | 45 |
| et_job_update . . . . .                | 46 |
| et_parquet_folders_done . . . . .      | 46 |

*et\_api\_run* 3

|                                          |    |
|------------------------------------------|----|
| et_parse_ami_filename . . . . .          | 47 |
| et_parse_camalien_filename . . . . .     | 48 |
| et_parse_edge_detections . . . . .       | 49 |
| et_parse_edge_track_detections . . . . . | 50 |
| et_parse_edge_tracks . . . . .           | 51 |
| et_process_edge_detections . . . . .     | 52 |
| et_process_edge_tracks . . . . .         | 53 |
| et_progress_start . . . . .              | 54 |
| et_sftp_batch . . . . .                  | 55 |
| et_sftp_connect . . . . .                | 56 |
| et_sftp_list_dated_subfolders . . . . .  | 57 |
| et_sftp_list_folders . . . . .           | 58 |
| parse_detections . . . . .               | 58 |
| parse_sftp_ls_la . . . . .               | 59 |

**Index** 60

---

|            |                                   |
|------------|-----------------------------------|
| et_api_run | <i>Run the erdatools REST API</i> |
|------------|-----------------------------------|

---

## 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 <code>plumber::pr_run()</code> . |

## Value

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

---

et\_build\_camalien\_images

*Build a CamAlien images data frame from an ERDA index*


---

## Description

Transforms a filtered ERDA index (files only) into the shape needed to write CamAlien on-board images into the camalien database data2 model. Every image attribute is parsed from the filename via `et_parse_camalien_filename()` — **no EXIF reads are performed**. One row is produced per image file.

## Usage

```
et_build_camalien_images(index, sessionid)
```

## Arguments

|           |                                                                                                                                                  |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| index     | Data frame as returned by <code>et_index_filter()</code> — must contain columns dir and name. Should be filtered to files only (no directories). |
| sessionid | Character UUID of the data2.session the images belong to (from <code>et_camalien_db_get_session()</code> ).                                      |

## Details

Filename tokens map to columns as follows: GT -> timestamp, CT -> computertime, lon/lat -> point position, expt -> exposuretime, gain, alt -> altitude, vel -> velocity, tag. CT is the onboard computer clock and GT the GPS-derived time; the backend treats the GPS time as the authoritative capture timestamp and keeps the computer clock in computertime (verified against the live camalien DB). The timestamps are UTC wall-clock in ISO 8601 basic format (e.g. 20250403T125240.225632) and are stamped with a trailing Z so the stored instant is independent of the database session timezone. Tokens absent from a filename become NA. All rows are stamped status = 1 (the initial data2.imagedata\_status, named pending); the backend PlantNet pipeline advances the status later.

Duplicate filenames in the index (the same file listed in more than one directory, e.g. a folder copied into a sibling on ERDA) are dropped with a warning, keeping the first occurrence. data2.imagefile.filename is globally unique, so a repeated filename is the same image; writing both would insert an orphaned data2.imagedata row whose imagefile insert is silently discarded by ON CONFLICT DO NOTHING.

## Value

Data frame with one row per image and columns filename, path, type, sessionid, lon, lat, timestamp, status, tag, computertime, exposuretime, gain, altitude, velocity.

## Examples

```
## Not run:
idx <- et_index_filter("erda_index.parquet", extensions = c("jpg", "tiff"))
imgs <- et_build_camalien_images(idx, sessionid = "3f2c8b1e-...-...")
```

```
## End(Not run)
```

---

```
et_build_classifications
```

*Build a classifications data frame for database insertion*

---

## Description

Produces up to two `data.classifications` rows per edge detection: an order classification (from the `order_*` fields) and a species classification (from the `species_*` fields). Detections are matched to their database UUIDs by joining on key, so `parsed_detect` and `detection_ids` must come from the same session (the edge key is unique per session only).

## Usage

```
et_build_classifications(
  parsed_detect,
  detection_ids,
  order_algorithm = "mcc24",
  species_algorithm = "uk-denmark-moths"
)
```

## Arguments

|                                |                                                                                                                                                         |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>parsed_detect</code>     | Tibble as returned by <code>et_parse_edge_detections()</code> .                                                                                         |
| <code>detection_ids</code>     | Data frame carrying detection UUIDs, as returned by <code>et_db_write_detections()</code> . Must contain columns <code>key</code> and <code>id</code> . |
| <code>order_algorithm</code>   | Character. Algorithm slug stamped on order classifications. Default "mcc24" (matches the 2044 backfill).                                                |
| <code>species_algorithm</code> | Character. Algorithm slug stamped on species classifications. Default "uk-denmark-moths" (matches the 2044 backfill).                                   |

## Details

Rows whose label is NA/empty are skipped. Following the database's own backfill (migration 2044), the single edge aboveTH flag is the order threshold, so the order row carries it as `abovethreshold` and the species row's `abovethreshold` is NA (SQL NULL).

## Value

A data frame with columns `detectionid`, `taxonlevel`, `label`, `labelid`, `score`, `abovethreshold`, `algorithm`.

## Examples

```
## Not run:
parsed <- et_parse_edge_detections("results/detect/20250508.csv")
det_ids <- et_db_write_detections(con, et_build_detections(parsed, si_ids))
classifications <- et_build_classifications(parsed, det_ids)

## End(Not run)
```

---

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. <code>c("Denmark" = "5", "Sweden" = "11")</code> ). 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, Trap type, Comments.

## Value

A `tibble::tibble()` with columns `year`, `code`, `locationname`, `elevation`, `latitude`, `longitude`, `power`, `trapserialno`, `trapstatus`, `traptype`, `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\_detection\_tracks

*Build a detection-track junction data frame for database insertion*

---

## Description

Joins parsed track-detail rows (which map detection key to a track edge\_id) to detection UUIDs (by key) and track UUIDs (by edge\_id), producing data.detection\_tracks rows. Detections whose key is absent from the track-detail file, or whose track has no UUID, produce no row.

## Usage

```
et_build_detection_tracks(
  trs_parsed,
  detection_ids,
  track_ids,
  algorithm = "tracy",
  algorithmversion = "luke"
)
```

## Arguments

|                  |                                                                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| trs_parsed       | Tibble as returned by <a href="#">et_parse_edge_track_detections()</a> . Must contain columns track_edge_id and key.                                    |
| detection_ids    | Data frame carrying detection UUIDs, as returned by <a href="#">et_db_write_detections()</a> . Must contain columns key and id.                         |
| track_ids        | Data frame carrying track UUIDs, as returned by <a href="#">et_db_get_tracks()</a> . Must contain columns edge_id, id, algorithm, and algorithmversion. |
| algorithm        | Character. Tracker algorithm name. Must be non-empty. Default "tracy".                                                                                  |
| algorithmversion | Character. Tracker algorithm version. Must be non-empty. Default "luke".                                                                                |

## Details

algorithm and algorithmversion are denormalised onto every junction row and **must equal the referenced track's identity** — the composite foreign key (trackid, algorithm, algorithmversion) -> data.tracks fails otherwise. Only tracks whose identity matches these arguments are considered when resolving trackid.

## Value

A data frame with columns detectionid, trackid, algorithm, algorithmversion.

## Examples

```
## Not run:
trs <- et_parse_edge_track_detections("results/tracks/20250508TRS.csv")
track_ids <- et_db_get_tracks(con, sessionid)
links <- et_build_detection_tracks(trs, det_ids, track_ids)

## End(Not run)
```

---

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| et_build_detections | <i>Build a detections data frame for database insertion</i> |
|---------------------|-------------------------------------------------------------|

---

## Description

Transforms the output of `et_parse_edge_detections()` into the `data.detections` schema. Each edge detection becomes one row: the source image is resolved by joining the detection filename (as a basename) to `sourceimage_ids`, and the bounding box is rounded to integer pixels. The `contour`, `cropurl`, and `contour_geom` columns are left NULL (the edge pipeline provides no polygon). `key` and `detect_id` are carried through for the downstream classifications and detection-track joins.

## Usage

```
et_build_detections(parsed_detect, sourceimage_ids)
```

## Arguments

|                              |                                                                                                                                                          |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>parsed_detect</code>   | Tibble as returned by <code>et_parse_edge_detections()</code> .                                                                                          |
| <code>sourceimage_ids</code> | Data frame with columns <code>filename</code> (basename) and <code>id</code> (source image UUID), as returned by <code>et_db_get_sourceimages()</code> . |

## Details

Detections whose filename has no matching source image are dropped (a warning reports the count).

**Value**

A data frame with columns sourceimageid, bbox\_x1, bbox\_y1, bbox\_x2, bbox\_y2 (integer), plus key and detect\_id for downstream joins.

**Examples**

```
## Not run:
parsed <- et_parse_edge_detections("results/detect/20250508.csv")
si_ids <- et_db_get_sourceimages(con, "MB050#2025")
detections <- et_build_detections(parsed, si_ids)

## End(Not run)
```

---

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| et_build_sessions | <i>Build a sessions data frame from grouped images</i> |
|-------------------|--------------------------------------------------------|

---

**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 <code>et_group_motion_images()</code> or <code>et_group_snapshot_images()</code> — 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 <code>et_index_filter()</code> — 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 | <i>Build a tracks data frame for database insertion</i> |
|-----------------|---------------------------------------------------------|

---

## 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 = "tracy",
  algorithmversion = "luke"
)

```

## Arguments

|                               |                                                                                                        |
|-------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>parsed_tracks</code>    | Tibble as returned by <code>et_parse_edge_tracks()</code> .                                            |
| <code>sessionid</code>        | Character. Session UUID to associate with all tracks.                                                  |
| <code>algorithm</code>        | Character. Tracker algorithm name stamped on every track. Must be a non-empty string. Default "tracy". |
| <code>algorithmversion</code> | Character. Tracker algorithm version. Must be a non-empty string. Default "luke".                      |

## Details

`algorithm` and `algorithmversion` are part of the `data.tracks` uniqueness key (`sessionid`, `edge_id`, `algorithm`, `algorithmversion`) which is `NULLS DISTINCT` (migration 2048). A `NULL`/empty identity would defeat the upsert dedupe and insert duplicate tracks on re-ingest, so both are required to be non-empty.

The tracker identity is named `tracy luke`: `tracy` is the tracker, and the version `luke` marks the on-edge tracker that predates `kylo`.

## 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_camalien_db_connect
```

*Connect to the camalien PostgreSQL database*

---

## Description

Wrapper around `DBI::dbConnect()` using `RPostgres::Postgres()`, parameterised by the `CAMALIEN_*` environment variables. This is intentionally **separate** from `et_db_connect()` (which targets the AMI database) so CamAlien writers can never accidentally connect to the wrong database — always call this function explicitly in CamAlien code paths (including inside mirai job bodies).

## Usage

```
et_camalien_db_connect(
  host = Sys.getenv("CAMALIEN_HOST"),
  port = 5432L,
  dbname = Sys.getenv("CAMALIEN_DB", unset = "camalien"),
  user = Sys.getenv("CAMALIEN_USER"),
  password = Sys.getenv("CAMALIEN_PASSWORD"),
  sslmode = "verify-full",
  sslrootcert = Sys.getenv("CAMALIEN_SSLROOTCERT")
)
```

## Arguments

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| host        | Database hostname. Defaults to CAMALIEN_HOST env var.                          |
| port        | Database port. Defaults to 5432.                                               |
| dbname      | Database name. Defaults to CAMALIEN_DB env var or "camalien".                  |
| user        | Database user. Defaults to CAMALIEN_USER env var.                              |
| password    | Database password. Defaults to CAMALIEN_PASSWORD env var.                      |
| sslmode     | SSL mode for the connection. Defaults to "verify-full".                        |
| sslrootcert | Path to the root CA certificate. Defaults to the CAMALIEN_SSLROOTCERT env var. |

## Details

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

**Value**

A DBI connection object.

**Examples**

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

## End(Not run)
```

---

et\_camalien\_db\_diff\_images

*Filter out CamAlien images already present in the database*

---

**Description**

Given built CamAlien image rows, returns only those whose filename is not already in data2.imagefile. data2.imagefile.filename is globally unique (uq\_v2\_imagefile\_filename), so it is the idempotency key — no session or partner scoping is required.

**Usage**

```
et_camalien_db_diff_images(dbcon, df, batch_size = 5000L)
```

**Arguments**

|            |                                                                                             |
|------------|---------------------------------------------------------------------------------------------|
| dbcon      | A DBI connection (from <code>et_camalien_db_connect()</code> ).                             |
| df         | Data frame with at least a filename column (from <code>et_build_camalien_images()</code> ). |
| batch_size | Integer. Filenames per lookup query. Default 5000L.                                         |

**Value**

The subset of df whose filenames are not yet in the database.

**Examples**

```
## Not run:
new <- et_camalien_db_diff_images(con, imgs)

## End(Not run)
```

---

et\_camalien\_db\_get\_partner

*Resolve a CamAlien partner id (project-scoped)*


---

## Description

Looks up a `data.partner.id` (UUID) in the camalien database, scoped to a project via `projectSlug`. camalien-db moved partner uniqueness to (`storagePath`, `projectSlug`), so the lookup is always project-scoped. This is a pure lookup — it never creates partners and errors if none matches.

## Usage

```
et_camalien_db_get_partner(
  dbcon,
  partner,
  project_slug = "onestop",
  by = c("name", "partnerkey", "storagepath")
)
```

## Arguments

|                           |                                                                                                              |
|---------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>dbcon</code>        | A DBI connection (from <code>et_camalien_db_connect()</code> ).                                              |
| <code>partner</code>      | Character. The partner identifier to match.                                                                  |
| <code>project_slug</code> | Character. Project slug to scope by. Defaults to "onestop" (the CamAlien on-board project).                  |
| <code>by</code>           | Character. Which column partner matches — one of "name", "partnerkey", or "storagepath". Defaults to "name". |

## Value

Character partner UUID.

## Examples

```
## Not run:
con <- et_camalien_db_connect()
pid <- et_camalien_db_get_partner(con, "Aarhus University")

## End(Not run)
```

---

et\_camalien\_db\_get\_session

*Look up a CamAlien session id (lookup only)*


---

### Description

Resolves a `data2.session.id` (UUID) by (`partnerid`, `name`, `year`). This is a pure lookup — sessions are created by the backend, so it errors if no session matches, and also errors if more than one matches (there is no unique constraint on the triple; pass a session UUID directly in that case).

### Usage

```
et_camalien_db_get_session(dbcon, partnerid, name, year)
```

### Arguments

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <code>dbcon</code>     | A DBI connection (from <code>et_camalien_db_connect()</code> ).           |
| <code>partnerid</code> | Character partner UUID (from <code>et_camalien_db_get_partner()</code> ). |
| <code>name</code>      | Character session name.                                                   |
| <code>year</code>      | Integer session year.                                                     |

### Value

Character session UUID.

### Examples

```
## Not run:
sid <- et_camalien_db_get_session(con, pid, "denmark", 2025L)

## End(Not run)
```

---

et\_camalien\_db\_write\_images

*Write CamAlien images to the camalien database (data2)*


---

### Description

Writes new CamAlien on-board image records into the `data2` model in a single transaction, in FK-safe order: `data2.imagedata` (with a client-generated UUID id — the column has no DB default — and a PostGIS point built from lon/lat), then `data2.additionalImagedata`, then `data2.imagefile`. The `data2.capturepoint` row is created automatically by the `trg_insert_capturepoint` trigger and is never written here.

**Usage**

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

**Arguments**

|            |                                                                                                                                                                                                          |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dbcon      | A DBI connection (from <code>et_camalien_db_connect()</code> ).                                                                                                                                          |
| df         | Data frame from <code>et_build_camalien_images()</code> — must contain columns filename, path, type, sessionid, lon, lat, timestamp, status, tag, computertime, exposureetime, gain, altitude, velocity. |
| batch_size | Integer. Rows per INSERT statement. Default 250L.                                                                                                                                                        |

**Details**

data2.imagefile uses ON CONFLICT (filename) DO NOTHING as a safety net; call `et_camalien_db_diff_images()` first so only genuinely new images are written. When lon/lat are NA the point is inserted as SQL NULL.

Duplicate filenames within df are an error: data2.imagefile.filename is globally unique, so the second copy's imagefile insert would be silently dropped by ON CONFLICT DO NOTHING, leaving its data2.imagedata row orphaned. `et_build_camalien_images()` deduplicates the index up front.

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

**Value**

The number of images written (invisibly).

**Examples**

```
## Not run:
con <- et_camalien_db_connect()
on.exit(DBI::dbDisconnect(con))
et_camalien_db_write_images(con, et_camalien_db_diff_images(con, imgs))

## End(Not run)
```

---

```
et_check_stray_deployments
```

*Check whether deployments exist for years found in stray images*

---

**Description**

After filtering stray images with `et_filter_stray_sourceimages()` or `et_filter_stray_images()`, this function examines the removed images to determine which years they belong to and whether matching deployments exist in the database. Emits informative cli messages and returns a summary data frame.

**Usage**

```
et_check_stray_deployments(removed, deployment_id, dbcon)
```

**Arguments**

|               |                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| removed       | Data frame of removed images from <code>et_filter_stray_sourceimages()</code> or <code>et_filter_stray_images()</code> . Must contain a timestamp column (character or POSIXct). |
| deployment_id | Character. The current deployment ID (e.g. "IT1#2023").                                                                                                                          |
| dbcon         | A DBI connection object.                                                                                                                                                         |

**Details**

The lookup uses the projectid, partnerid, and code from the current deployment, substituting each stray year.

**Value**

Data frame with columns:

**year** Integer. Year extracted from stray image timestamps.

**n\_images** Integer. Number of stray images in that year.

**deployment\_id** Character. The deployment ID if it exists, or NA\_character\_.

**deployment\_exists** Logical. Whether a deployment was found.

Returns a zero-row data frame if removed is empty.

**Examples**

```
## Not run:
stray <- et_filter_stray_sourceimages(si, year = 2023)
if (nrow(stray$removed) > 0L) {
  info <- et_check_stray_deployments(stray$removed, "IT1#2023", con)
}

## 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/code/ (no year or partner level; trap codes are unique within a project and images are organised in date-based subdirectories)

For standalone traps, a partner folder of "root" (or an unknown partner) is a sentinel meaning the deployment sits directly under the year, so the partner level is omitted (e.g. /vibes/2026/VIB01/).

**Value**

Character string with the remote path, e.g. "/ias/2023/italy/IT1/" for standalone or "/ias/BI025/" 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 <a href="#">et_build_deployments()</a> ). |

**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 <a href="#">et_db_connect()</a> ). 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 <a href="#">et_build_sourceimages()</a> ).                                   |
| batch_size | Integer. Number of filenames per query. Default 5000L.                                                                                           |
| n_workers  | Integer. Number of parallel mirai workers. Default 1L. See <a href="#">et_db_write_sourceimages()</a> 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 trapttype (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 <a href="#">et_db_connect()</a> ). |
| 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_get_sourceimages`*Get source images for a deployment*

---

**Description**

Queries data.sourceimages for all images belonging to a deployment, returning the filename (basename) and UUID of each. Used to resolve edge detection filenames to source-image UUIDs when writing data.detections.

**Usage**

```
et_db_get_sourceimages(dbcon, deploymentid)
```

**Arguments**

|              |                                                               |
|--------------|---------------------------------------------------------------|
| dbcon        | A DBI connection object (from <code>et_db_connect()</code> ). |
| deploymentid | Character. Deployment ID (e.g. "MB050#2025").                 |

**Value**

Data frame with columns: id (character, UUID) and filename (character, basename). Zero rows if no source images exist.

**Examples**

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
si_ids <- et_db_get_sourceimages(con, "MB050#2025")

## End(Not run)
```

---

|                  |                                 |
|------------------|---------------------------------|
| et_db_get_tracks | <i>Get tracks for a session</i> |
|------------------|---------------------------------|

---

### Description

Queries data.tracks for all tracks belonging to a session, returning the track UUID keyed by (edge\_id, algorithm, algorithmversion). Used to resolve edge track IDs to track UUIDs when writing the data.detection\_tracks junction (the writers `et_db_write_tracks()` return only a row count).

### Usage

```
et_db_get_tracks(dbcon, sessionid)
```

### Arguments

|           |                                                               |
|-----------|---------------------------------------------------------------|
| dbcon     | A DBI connection object (from <code>et_db_connect()</code> ). |
| sessionid | Character. Session UUID.                                      |

### Value

Data frame with columns: id (character, UUID), edge\_id (integer), algorithm (character), algorithmversion (character). Zero rows if no tracks exist.

### Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
track_ids <- et_db_get_tracks(con, "abc-123")

## End(Not run)
```

---

|                        |                                              |
|------------------------|----------------------------------------------|
| et_db_list_deployments | <i>List deployment IDs from the database</i> |
|------------------------|----------------------------------------------|

---

### 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, traptype = 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.                                                                     |
| traptype  | Character or NULL. Trap type to filter by, one of "standalone", "iot", "lepisense", or "mothbox". If NULL, deployments of every trap type 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\_classifications

*Write classifications to the database*

---

**Description**

Performs batch INSERT into data.classifications. On conflict with uq\_classifications\_detectionid\_taxonlevel\_ (detectionid, taxonlevel, algorithm) the row is updated (DO UPDATE SET label, labelid, score, abovethreshold), so re-ingest refreshes scores. embeddingpath is left NULL (the edge pipeline provides none).

**Usage**

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

**Arguments**

|            |                                                                                                                                                                          |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dbcon      | A DBI connection object (from <a href="#">et_db_connect()</a> ).                                                                                                         |
| df         | Data frame with columns: detectionid, taxonlevel, label, labelid, score, abovethreshold, algorithm. Typically the output of <a href="#">et_build_classifications()</a> . |
| 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))
cls <- et_build_classifications(parsed, det_ids)
et_db_write_classifications(con, cls)

## 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 <a href="#">et_db_connect()</a> ).                                                                                             |
| 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\_detection\_tracks

*Write detection-track links to the database*

---

**Description**

Performs batch INSERT into data.detection\_tracks. On conflict with the primary key (detectionid, trackid) the row is skipped (DO NOTHING), making re-ingest idempotent. The algorithm/algorithmversion on each row must equal the referenced track's identity or the composite foreign key to data.tracks fails.

**Usage**

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

**Arguments**

|            |                                                                                                                                                   |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| dbcon      | A DBI connection object (from <a href="#">et_db_connect()</a> ).                                                                                  |
| df         | Data frame with columns: detectionid, trackid, algorithm, algorithmversion. Typically the output of <a href="#">et_build_detection_tracks()</a> . |
| 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))
links <- et_build_detection_tracks(trs, det_ids, track_ids)
et_db_write_detection_tracks(con, links)

## End(Not run)
```

---

et\_db\_write\_detections

*Write detections to the database*


---

## Description

Performs batch INSERT into data.detections. Each detection receives a UUID generated by `uuid_generate_v4()` in PostgreSQL (the `id` column has no default). On conflict with `uq_detections_sourceimage_bbox` (`sourceimageid`, `bbox_x1`, `bbox_y1`, `bbox_x2`, `bbox_y2`) the row is skipped (DO NOTHING), making re-ingest idempotent. After writing, the detection UUIDs are read back and matched onto the input rows by (`sourceimageid`, `bbox_x1`, `bbox_y1`, `bbox_x2`, `bbox_y2`), mirroring `et_db_write_sessions()`.

## Usage

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

## Arguments

|                         |                                                                                                                                                                                                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dbcon</code>      | A DBI connection object (from <code>et_db_connect()</code> ).                                                                                                                                                                                                                                                         |
| <code>df</code>         | Data frame with columns: <code>sourceimageid</code> , <code>bbox_x1</code> , <code>bbox_y1</code> , <code>bbox_x2</code> , <code>bbox_y2</code> . Typically the output of <code>et_build_detections()</code> . Any extra columns (e.g. <code>key</code> , <code>detect_id</code> ) are preserved in the return value. |
| <code>batch_size</code> | Integer. Number of rows per INSERT statement. Default 250L.                                                                                                                                                                                                                                                           |

## Details

`contour`, `cropurl`, and `contour_geom` are left NULL (the edge pipeline provides no polygon); `area_px` is a generated column and is never written.

## Value

The input data frame with an `id` column (character UUID) populated from the database. Carries an `n_written` attribute: the number of rows actually inserted (rows skipped by the DO NOTHING conflict clause are not counted).

## Examples

```
## Not run:
con <- et_db_connect()
on.exit(DBI::dbDisconnect(con))
det <- et_build_detections(parsed, si_ids)
det_ids <- et_db_write_detections(con, det)

## 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 <a href="#">et_db_connect()</a> ). 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 <a href="#">et_build_sessions()</a> .                               |
| batch_size | Integer. Number of rows per INSERT statement. Default 250L.                                                                                                        |
| n_workers  | Integer. Number of parallel mirai workers. Default 1L. See <a href="#">et_db_write_sourceimages()</a> 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 <a href="#">et_db_connect()</a> ). 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 <a href="#">mirai::mirai_map()</a> and each worker opens and closes its own DB connection. Requires the <b>mirai</b> package and an installed (not just loaded) <b>erdatools</b> . |

**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_tracking_tasks
```

*Write tracking-task status rows to the database*

---

**Description**

Performs batch upsert into `data.tracking_tasks`, the session-level tracking work queue. Used by the IoT path to record that a session's tracks were produced on the edge device: each row is written with `status = "done"` and the tracker `algorithm/algorithmversion`, which mark the edge provenance. The unique key is `(sessionid, algorithm, algorithmversion)` (constraint `uq_tracking_tasks_session_algorithm`), so on conflict the existing row's status is updated (DO UPDATE), making re-ingest idempotent. Every row needs a `workerid`—the `tracking_tasks` CHECK requires a non-NULL `workerid` for any non-pending status. Edge-tracked sessions have no server-side worker, so callers pass a fixed virtual edge-worker id (see `.et_edge_workerid` in `erda_edge.R`); `workerid` has no foreign key.

Usage

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

Arguments

|            |                                                                                                                                                                                                                                     |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dbcon      | A DBI connection object (from <code>et_db_connect()</code> ).                                                                                                                                                                       |
| df         | Data frame with columns: sessionid, status, algorithm, algorithmversion, workerid. status must be a valid data.tracking_task_status value (pending/in_progress/done/failure). workerid must be non-NULL for any non-pending status. |
| 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))
tasks <- data.frame(
  sessionid = "abc-123",
  status = "done",
  algorithm = "tracy",
  algorithmversion = "luke",
  workerid = "f2491a34-c18a-5188-984b-09092a9aaa55"
)
et_db_write_tracking_tasks(con, tasks)

## End(Not run)
```

---

|                    |                                     |
|--------------------|-------------------------------------|
| et_db_write_tracks | <i>Write tracks to the database</i> |
|--------------------|-------------------------------------|

---

Description

Performs batch INSERT into data.tracks. On conflict with the uq\_tracks\_session\_edge\_algorithm constraint (sessionid, edge\_id, algorithm, algorithmversion) the row is skipped (DO NOTHING), making re-ingest idempotent. 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 <code>et_db_connect()</code> ).                                                                      |
| df         | Data frame with columns: sessionid, edge_id, algorithm, algorithmversion. Typically the output of <code>et_build_tracks()</code> . |
| 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*

---

### Description

Removes rows from a grouped image data frame where the timestamp year does not match the deployment year exactly.

### Usage

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

### Arguments

|          |                                                                                                                                                                             |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| grouped  | Data frame as returned by <a href="#">et_group_motion_images()</a> or <a href="#">et_group_snapshot_images()</a> . 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*

---

**Description**

Removes rows from a sourceimages data frame where the timestamp year does not match the deployment year exactly. 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 <a href="#">et_build_sourceimages()</a> . 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. <code>"/storage/onestop/2025/"</code> ). Leading and trailing slashes are stripped before joining. |
| country    | Country subdirectory name (e.g. <code>"portugal"</code> ).                                                                                                                   |
| folder     | Dated subfolder name (e.g. <code>"2025_07_10-11_10"</code> ).                                                                                                                |
| filename   | Image file name (e.g. <code>"img.jpg"</code> ).                                                                                                                              |
| http_base  | Base URL of the ERDA share-redirect endpoint. Defaults to <code>"https://anon.erda.au.dk/share_redirect"</code> .                                                            |

**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

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>conn</code>             | Connection object returned by <code>et_sftp_connect()</code> .                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>remote_path</code>      | Remote root path to index.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>index_path</code>       | Local path for the output parquet file.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>checkpoint_every</code> | Integer. Rewrite the parquet every this many directories. Default 25L.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>n_workers</code>        | 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 <b>mirai</b> package and an installed <b>erdatools</b> . Safe to call from inside an existing mirai daemon (the workers form a nested pool). Each concurrent worker counts toward ERDA's per-sharelink limit of 16 active sessions; scale back if listings stall. |

## Details

On a fresh run, remote\_path must exist: a missing root aborts with an error instead of producing an empty index.

**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 <code>et_index_dir()</code> , or a character string of length 1 giving the path to a parquet file.                                                                                             |
| extensions | Character vector of file extensions to keep, e.g. <code>c("jpg", "jpeg")</code> or <code>c(".tif")</code> . 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 | <i>Detect additions and removals in an existing ERDA index</i> |
|-----------------|----------------------------------------------------------------|

---

## 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 <code>et_sftp_connect()</code> .                       |
| 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 | <i>List all jobs</i> |
|-------------|----------------------|

---

**Description**

Syncs status from background process handles before returning.

**Usage**

```
et_job_list()
```

**Value**

A data frame with columns: id, type, status, stage, created\_at, updated\_at, last\_progress\_at. stage and last\_progress\_at are NA for jobs that have not reported progress. Returns an empty data frame if no jobs exist.

**Examples**

```
et_job_list()
```

---

|                      |                                      |
|----------------------|--------------------------------------|
| et_job_progress_file | <i>Path of a job's progress file</i> |
|----------------------|--------------------------------------|

---

**Description**

Background workers cannot write to the in-memory job store, so they report progress by writing a small JSON file that `et_job_get()` reads. This returns the path for a given job ID, under a per-process directory in `base::tempdir()`. Pass it to the worker and call `et_progress_start()` there.

**Usage**

```
et_job_progress_file(id)
```

**Arguments**

id                      Character. Job ID.

**Value**

Character string: absolute path of the progress file. The file itself only exists once the worker has reported progress.

**Examples**

```
et_job_progress_file("index_full_20260101_120000_abc123")
```

---

|                    |                                                  |
|--------------------|--------------------------------------------------|
| et_job_set_process | <i>Attach a background mirai handle to a job</i> |
|--------------------|--------------------------------------------------|

---

**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 | <i>Update a job's status</i> |
|---------------|------------------------------|

---

**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 | <i>Check which folders already have a parquet file</i> |
|-------------------------|--------------------------------------------------------|

---

**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)
```



**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\_56.78). 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(filenamees)
```

**Arguments**

`filenamees`      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_track_detections
      Parse edge track detail CSV
```

---

### Description

Reads a track detail CSV produced by IoT AMI trap edge processing (e.g. results/tracks/YYYYMMDDTRS.csv).  
 One row per detection that belongs to a track; supplies the key -> track edge\_id map used to build the data.detection\_tracks junction.

### Usage

```
et_parse_edge_track_detections(path)
```

### Arguments

path                      Character. Path to the track detail CSV file.

### Value

A `tibble::tibble()` with columns:

**track\_edge\_id** Integer. Track ID from the edge processor (joins to data.tracks.edge\_id).  
**key** Integer. Detection key (joins to the detection CSV key).  
**timestamp** POSIXct (UTC). Detection time.  
**confidence** Double. Species classification confidence (percent).  
**valid** Logical. Whether confidence exceeds the threshold.  
**order** Character. Taxonomic order label.  
**species** Character. Species-level label.  
**xc, yc** Integer. Bounding-box centre (pixels).  
**x1, y1** Integer. Top-left corner of the bounding box (pixels).  
**width, height** Integer. Bounding-box dimensions (pixels).  
**x2, y2** Integer. Bottom-right corner (x1 + width, y1 + height).  
**image** Character. Source image filename (already a basename).

## Examples

```
## Not run:
trs <- et_parse_edge_track_detections("results/tracks/20250508TRS.csv")

## End(Not run)
```

---

|                      |                                     |
|----------------------|-------------------------------------|
| et_parse_edge_tracks | <i>Parse edge track summary CSV</i> |
|----------------------|-------------------------------------|

---

## 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\_detections

*Download, parse, and write edge detections to the database*


---

## Description

Finds detection CSVs (results/detect/YYYYMMDD.csv) and track-detail CSVs (results/tracks/YYYYMMDDTRS.csv) in an ERDA index, downloads each via SFTP, and writes data.detections, data.classifications, and the data.detection\_tracks junction. Source images, sessions, and tracks must already exist in the database (written upstream); this function fetches their UUIDs via [et\\_db\\_get\\_sourceimages\(\)](#) and [et\\_db\\_get\\_tracks\(\)](#).

## Usage

```
et_process_edge_detections(
  conn,
  dbcon,
  idx_raw,
  sessions,
  deploymentid,
  algorithm = "tracy",
  algorithmversion = "luke",
  order_algorithm = "mcc24",
  species_algorithm = "uk-denmark-moths"
)
```

## Arguments

|                   |                                                                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| conn              | SFTP connection object from <a href="#">et_sftp_connect()</a> .                                                                                        |
| dbcon             | A DBI connection object.                                                                                                                               |
| idx_raw           | Data frame. Raw ERDA index (as returned by <a href="#">et_index_dir()</a> ).                                                                           |
| sessions          | Data frame. Sessions with id, date, and type columns (as returned by <a href="#">et_db_get_sessions()</a> or <a href="#">et_db_write_sessions()</a> ). |
| deploymentid      | Character. Deployment ID, used to fetch source-image UUIDs.                                                                                            |
| algorithm         | Character. Tracker algorithm name for the junction rows. Must match the tracks written upstream. Default "tracy".                                      |
| algorithmversion  | Character. Tracker algorithm version for the junction rows. Must match the tracks written upstream. Default "luke".                                    |
| order_algorithm   | Character. Algorithm slug for order classifications. Default "mcc24".                                                                                  |
| species_algorithm | Character. Algorithm slug for species classifications. Default "uk-denmark-moths".                                                                     |

**Details**

Writes happen in foreign-key-safe order: detections, then classifications, then the detection-track junction. Each detection CSV is matched to its track-detail CSV and motion session by the session date encoded in the filename.

**Value**

A named list with integer counts: detections, classifications, and detection\_tracks (rows written).

---

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,
  algorithm = "tracy",
  algorithmversion = "luke"
)
```

**Arguments**

|                  |                                                                                                                                             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| conn             | SFTP connection object from <a href="#">et_sftp_connect()</a> .                                                                             |
| dbcon            | A DBI connection object.                                                                                                                    |
| idx_raw          | Data frame. Raw ERDA index (as returned by <a href="#">et_index_dir()</a> ).                                                                |
| sessions         | Data frame. Sessions with id, date, and type columns (as returned by <a href="#">et_db_write_sessions()</a> ).                              |
| algorithm        | Character. Tracker algorithm name stamped on every track. Must be non-empty. Default "tracy". Passed to <a href="#">et_build_tracks()</a> . |
| algorithmversion | Character. Tracker algorithm version. Must be non-empty. Default "luke". Passed to <a href="#">et_build_tracks()</a> .                      |

**Value**

Integer. Total number of track rows written.

---

|                   |                                                     |
|-------------------|-----------------------------------------------------|
| et_progress_start | <i>Report job progress from a background worker</i> |
|-------------------|-----------------------------------------------------|

---

## Description

et\_progress\_start() points the current process at a job's progress file (from et\_job\_progress\_file()); call it at the top of a mirai::mirai() job body. et\_progress() then records the job's current stage and counters in that file, where et\_job\_get() picks them up and /index/status/{id} reports them as stage, progress and last\_progress\_at. et\_progress\_stop() detaches the reporter again.

## Usage

```
et_progress_start(path)

et_progress_stop()

et_progress(stage, ..., interval = 2)

et_progress_read(path)
```

## Arguments

|          |                                                                                                                              |
|----------|------------------------------------------------------------------------------------------------------------------------------|
| path     | Character. Progress file path, or NULL to stop reporting.                                                                    |
| stage    | Character. Short name of the current stage, e.g. "index".                                                                    |
| ...      | Named counters describing progress within the stage, e.g. dirs_processed = 120L. Replaces the counters of the previous call. |
| interval | Numeric. Minimum seconds between writes within one stage.                                                                    |

## Details

Long-running functions (et\_index\_dir(), et\_index\_update(), the edge processors and the database writers) call et\_progress() themselves, so a job only needs to mark the stages between them. Outside a reporting job (no et\_progress\_start()), all calls are no-ops.

Writes are throttled to one every interval seconds unless the stage changes, and each write is atomic (temp file + rename), so a reader never sees a partial file.

## Value

et\_progress\_start(), et\_progress\_stop() and et\_progress(): NULL, invisibly. et\_progress\_read(): a list with stage, progress (named list of counters, or NULL), last\_progress\_at (POSIXct) and pid (the worker's process ID), or NULL if the file does not exist or cannot be parsed.

**Examples**

```
f <- tempfile(fileext = ".json")
et_progress_start(f)
et_progress("index", dirs_processed = 10L, files = 250L)
et_progress_read(f)
et_progress_stop()
```

---

|               |                                  |
|---------------|----------------------------------|
| et_sftp_batch | <i>Run an SFTP batch session</i> |
|---------------|----------------------------------|

---

**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, timeout = 0)
```

**Arguments**

|          |                                                                                                                                                                                   |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| conn     | Connection object returned by <code>et_sftp_connect()</code> .                                                                                                                    |
| commands | Character vector of SFTP commands, passed to the session via stdin.                                                                                                               |
| stdout   | Passed to <code>base::system2()</code> . TRUE (default) captures output as a character vector.                                                                                    |
| stderr   | Passed to <code>base::system2()</code> . FALSE (default) discards stderr.                                                                                                         |
| timeout  | Numeric. Seconds after which the session is killed, passed to <code>base::system2()</code> . 0 (default) means no limit. A killed session returns exit status 124 with a warning. |

**Value**

Character vector of stdout lines (when `stdout = TRUE`), or the integer exit code (when `stdout = FALSE`). With `stdout = TRUE`, a non-zero exit status is attached as the "status" attribute.

**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 -l /storage/2025/"))

## End(Not run)
```

---

|                 |                                              |
|-----------------|----------------------------------------------|
| et_sftp_connect | <i>Connect to ERDA via SSH ControlMaster</i> |
|-----------------|----------------------------------------------|

---

### 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 <code>base::tempdir()</code> . |

### 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:

```
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 <code>et_sftp_connect()</code> . |
| 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 -l <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:

```
{"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).

---

|                               |                                     |
|-------------------------------|-------------------------------------|
| <code>parse_sftp_ls_la</code> | <i>Parse raw SFTP ls -la output</i> |
|-------------------------------|-------------------------------------|

---

**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>`. 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

base::basename(), 47, 57  
base::system2(), 55  
base::tempdir(), 44, 56  
  
cli::cli\_abort(), 56  
  
DBI::dbConnect(), 12, 18  
DBI::dbExecute(), 31  
  
et\_api\_run, 3  
et\_build\_camalien\_images, 4  
et\_build\_camalien\_images(), 13, 16  
et\_build\_classifications, 5  
et\_build\_classifications(), 27  
et\_build\_deployments, 6  
et\_build\_deployments(), 20  
et\_build\_detection\_tracks, 7  
et\_build\_detection\_tracks(), 29  
et\_build\_detections, 8  
et\_build\_detections(), 30  
et\_build\_sessions, 9  
et\_build\_sessions(), 31  
et\_build\_sourceimages, 10  
et\_build\_sourceimages(), 21, 36  
et\_build\_tracks, 11  
et\_build\_tracks(), 34, 53  
et\_camalien\_db\_connect, 12  
et\_camalien\_db\_connect(), 13–16  
et\_camalien\_db\_diff\_images, 13  
et\_camalien\_db\_diff\_images(), 16  
et\_camalien\_db\_get\_partner, 14  
et\_camalien\_db\_get\_partner(), 15  
et\_camalien\_db\_get\_session, 15  
et\_camalien\_db\_get\_session(), 4  
et\_camalien\_db\_write\_images, 15  
et\_check\_stray\_deployments, 16  
et\_db\_assign\_sessions, 17  
et\_db\_connect, 18  
et\_db\_connect(), 12, 18, 21, 24–34  
et\_db\_deployment\_path, 19  
et\_db\_diff\_deployments, 20  
et\_db\_diff\_deployments(), 28  
et\_db\_diff\_sourceimages, 21  
et\_db\_get\_deployment, 22  
et\_db\_get\_deployment\_id, 22  
et\_db\_get\_partner\_folder, 23  
et\_db\_get\_project\_folder, 24  
et\_db\_get\_sessions, 24  
et\_db\_get\_sessions(), 52  
et\_db\_get\_sourceimages, 25  
et\_db\_get\_sourceimages(), 8, 52  
et\_db\_get\_tracks, 26  
et\_db\_get\_tracks(), 7, 52  
et\_db\_list\_deployments, 26  
et\_db\_write\_classifications, 27  
et\_db\_write\_deployments, 28  
et\_db\_write\_deployments(), 6  
et\_db\_write\_detection\_tracks, 29  
et\_db\_write\_detections, 30  
et\_db\_write\_detections(), 5, 7  
et\_db\_write\_sessions, 31  
et\_db\_write\_sessions(), 30, 52, 53  
et\_db\_write\_sourceimages, 31  
et\_db\_write\_sourceimages(), 21, 31  
et\_db\_write\_tracking\_tasks, 32  
et\_db\_write\_tracks, 33  
et\_db\_write\_tracks(), 26, 53  
et\_exif\_tags, 34  
et\_filter\_media\_files, 35  
et\_filter\_stray\_images, 35  
et\_filter\_stray\_images(), 16, 17, 36  
et\_filter\_stray\_sourceimages, 36  
et\_filter\_stray\_sourceimages(), 16, 17  
et\_group\_motion\_images, 37  
et\_group\_motion\_images(), 9, 35  
et\_group\_snapshot\_images, 38  
et\_group\_snapshot\_images(), 9, 35  
et\_img\_url, 39  
et\_index\_dir, 40

`et_index_dir()`, 41, 42, 52–54  
`et_index_filter`, 41  
`et_index_filter()`, 4, 10, 37, 38  
`et_index_update`, 42  
`et_index_update()`, 54  
`et_job_create`, 43  
`et_job_get`, 43  
`et_job_get()`, 44, 45, 54  
`et_job_list`, 44  
`et_job_list()`, 45  
`et_job_progress_file`, 44  
`et_job_progress_file()`, 54  
`et_job_set_process`, 45  
`et_job_update`, 46  
`et_parquet_folders_done`, 46  
`et_parse_ami_filename`, 47  
`et_parse_ami_filename()`, 37, 38  
`et_parse_camalien_filename`, 48  
`et_parse_camalien_filename()`, 4  
`et_parse_edge_detections`, 49  
`et_parse_edge_detections()`, 5, 8  
`et_parse_edge_track_detections`, 50  
`et_parse_edge_track_detections()`, 7  
`et_parse_edge_tracks`, 51  
`et_parse_edge_tracks()`, 11, 53  
`et_process_edge_detections`, 52  
`et_process_edge_tracks`, 53  
`et_progress (et_progress_start)`, 54  
`et_progress_read (et_progress_start)`, 54  
`et_progress_start`, 54  
`et_progress_start()`, 44  
`et_progress_stop (et_progress_start)`, 54  
`et_sftp_batch`, 55  
`et_sftp_batch()`, 40, 59  
`et_sftp_connect`, 56  
`et_sftp_connect()`, 40, 42, 52, 53, 55, 57, 58  
`et_sftp_list_dated_subfolders`, 57  
`et_sftp_list_folders`, 58  
`et_sftp_list_folders()`, 57  
  
`mirai::mirai()`, 45, 54  
`mirai::mirai_map()`, 32  
  
`parse_detections`, 58  
`parse_sftp_ls_la`, 59  
`plumber::pr_run()`, 3  
  
`RPostgres::Postgres()`, 12, 18  
  
`tibble::tibble()`, 6, 49–51, 59