

# Package: aniread (via r-universe)

October 14, 2026

**Type** Package

**Title** An R package for reading and writing movement data

**Version** 0.7.0.9006

**Description** An R package for reading and writing movement data.

**License** MIT + file LICENSE

**URL** <https://animovement.dev/aniread/>,  
<https://github.com/animovement/aniread/>

**BugReports** <https://github.com/animovement/aniread/issues>

**Encoding** UTF-8

**Language** en-GB

**Depends** R (>= 4.1.0)

**Imports** anicore (>= 0.8.0.9009), cli, dplyr, janitor, lifecycle,  
rlang, stats, tidyr, tidyselect, utils, vroom

**Suggests** arrow, covr, curl, rhdf5, jsonlite, yaml, xml2, c3dr,  
testthat (>= 3.0.0), withr

**Additional\_repositories** <https://animovement.r-universe.dev>,  
<https://bioc.r-universe.dev>

**Config/testthat/edition** 3

**Config/Needs/website** animovement/animovementtemplate, knitr

**Roxygen** list(markdown = TRUE)

**Config/roxygen2/version** 8.1.0

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

**Date/Publication** 2026-10-14 15:01:49 UTC

**RemoteUrl** <https://github.com/animovement/aniread>

**RemoteRef** HEAD

**RemoteSha** 22974a73834211b76f9ef478220905e45d77ad26

## Contents

|                                 |           |
|---------------------------------|-----------|
| calibrate_trackball . . . . .   | 2         |
| detect_source . . . . .         | 3         |
| get_sample_data . . . . .       | 4         |
| get_supported_sources . . . . . | 7         |
| read_aniframe . . . . .         | 8         |
| read_animalta . . . . .         | 8         |
| read_anipose . . . . .          | 10        |
| read_bonsai . . . . .           | 12        |
| read_boris . . . . .            | 13        |
| read_c3d . . . . .              | 14        |
| read_custom . . . . .           | 15        |
| read_dataset . . . . .          | 16        |
| read_deeplabcut . . . . .       | 19        |
| read_fasttrack . . . . .        | 20        |
| read_fictrac . . . . .          | 21        |
| read_freemocap . . . . .        | 22        |
| read_idtracker . . . . .        | 25        |
| read_lightningpose . . . . .    | 27        |
| read_movement . . . . .         | 28        |
| read_octron . . . . .           | 29        |
| read_sleap . . . . .            | 31        |
| read_structure . . . . .        | 32        |
| read_trackball . . . . .        | 34        |
| read_trackmate . . . . .        | 36        |
| read_trex . . . . .             | 38        |
| write_aniframe . . . . .        | 40        |
| write_dataset . . . . .         | 41        |
| write_intracktive . . . . .     | 42        |
| <b>Index</b>                    | <b>45</b> |

---

|                     |                                                  |
|---------------------|--------------------------------------------------|
| calibrate_trackball | <i>Get calibration values for trackball data</i> |
|---------------------|--------------------------------------------------|

---

## Description

Reads a calibration recording and computes calibration values. Spin the ball a known number of times in one direction while recording.

## Usage

```
calibrate_trackball(
    path,
    ball_diameter,
    ball_rotations,
    col_dx = "x",
```

```
    col_dy = "y"
  )
```

### Arguments

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| path           | Path to calibration CSV file.                                       |
| ball_diameter  | Diameter of the trackball in desired output units (e.g., mm or cm). |
| ball_rotations | Number of complete rotations performed.                             |
| col_dx         | Column name for x-axis values.                                      |
| col_dy         | Column name for y-axis values.                                      |

### Value

A list with two elements:

- counts\_per\_rotation: Sensor counts per full rotation (for ball\_calibration in read\_trackball).
- calibration\_factor: Distance per sensor count (for anicore::convert\_unit\_space()).

### Examples

```
# Roll the ball a known number of turns, then convert sensor counts to cm
path <- system.file("extdata", "trackball_sensor_1.csv", package = "aniread")
calibrate_trackball(path, ball_diameter = 5, ball_rotations = 10)
```

---

|               |                                                  |
|---------------|--------------------------------------------------|
| detect_source | <i>Detect which source software wrote a file</i> |
|---------------|--------------------------------------------------|

---

### Description

Works out which of the supported source software formats a file is in, by looking at the file itself rather than trusting its name. This is what [read\\_dataset\(\)](#) uses when source = "auto".

Candidates are first narrowed to the sources registered for the file's suffix, then each candidate's detector inspects the file's contents. Exactly one match is required.

### Usage

```
detect_source(paths)
```

### Arguments

|       |                                                                                                                        |
|-------|------------------------------------------------------------------------------------------------------------------------|
| paths | Path to the file. If several paths are given (as <a href="#">read_trackball()</a> takes), only the first is inspected. |
|-------|------------------------------------------------------------------------------------------------------------------------|

## Details

DeepLabCut and LightningPose export structurally identical CSV files, so a file in that format matches both. Rather than guess, `detect_source()` returns the combined name "deeplabcut/lightningpose"; `read_dataset()` reads it with `read_deeplabcut()` - the parse is the same either way - and records the combined name as the source, so the ambiguity is preserved rather than silently resolved.

Detectors for HDF5, Parquet, XML and C3D files need an optional package (rhdf5, arrow, xml2, c3dr). When one is not installed its sources are skipped, and if nothing is detected the error names both the skipped sources and the packages that would have been consulted.

## Value

The source name, as it appears in `get_supported_sources()`, or the combined "deeplabcut/lightningpose".

## Writing a detector

A detector is a function of a single path returning TRUE when the file is of that source. It must be cheap - read a handful of lines, or an HDF5 index, never the whole file - and it need not be defensive: `detect_source()` runs each one with messages and warnings suppressed and treats an error as "did not match".

## See Also

`read_dataset()` to read the file, `get_supported_sources()` for the supported formats.

## Examples

```
detect_source(system.file("extdata", "sleap.analysis.csv", package = "aniread"))

# DeepLabCut and Lightning Pose write the same CSV layout
detect_source(system.file("extdata", "deeplabcut.csv", package = "aniread"))
```

---

|                 |                                      |
|-----------------|--------------------------------------|
| get_sample_data | <i>Download sample tracking data</i> |
|-----------------|--------------------------------------|

---

## Description

Downloads sample data for different animal tracking software and returns the path to the downloaded file. The function caches the data to avoid repeated downloads.

## Usage

```
get_sample_data(
  source,
  dataset = NULL,
  cache_dir = tempdir(),
  quiet = FALSE,
  list_datasets = FALSE
)
```

**Arguments**

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| source        | <p>Character string specifying either a source name, as <code>read_dataset()</code> names it, or a URL. Currently supported source names:</p> <ul style="list-style-type: none"> <li>• "animalta": Data from AnimalTA</li> <li>• "anipose": Mouse paw tracking data</li> <li>• "bonsai": Tracking data from Bonsai</li> <li>• "c3d": Motion capture in C3D (2 datasets): "example", a walking trial, and "sample-static", a static calibration trial</li> <li>• "deeplabcut": Mouse/animal tracking from DeepLabCut (4 datasets)</li> <li>• "fasttrack": Tracking from FastTrack</li> <li>• "fictrac": Fictrac sample data</li> <li>• "freemocap": FreeMoCap motion capture of a person doing star jumps (4 datasets), from movement's sample data, written by FreeMoCap's own saver. The default, "star-jump", is the 9-column <code>by_frame.csv</code> of v1.7.4 and later; "star-jump_v1.7" is the 8-column one of earlier versions; "star-jump_by_trajectory" and "star-jump_wide" are the recording's <code>by_trajectory.csv</code> and <code>mediapipe_body_3d_xyz.csv</code></li> <li>• "idtrackerai": Trajectories from idtracker.ai (2 datasets): the <code>.h5</code> export, and "trajectories_csv", the CSV export</li> <li>• "lightningpose": Mouse tracking from LightningPose (3 datasets). The default, "IBL-paw_EKS-left", is the Ensemble Kalman Smoother's output for one camera</li> <li>• "movement": netCDF files saved by the movement Python package (2 datasets). The default, "two-mice_octagon", has the dimension names movement uses since 0.17.0; "legacy-plural" is the same recording with the plural names of earlier versions</li> <li>• "octron": Segmentation tracking from OCTRON</li> <li>• "sleap": Animal tracking from SLEAP (4 datasets)</li> <li>• "trackball_bonsai": Two optical-flow sensors under a trackball, logged with Bonsai. Unpacks to one file per sensor and returns both paths, which <code>read_trackball()</code> reads as one recording</li> <li>• "trackmate": Cell tracking from TrackMate (3 datasets)</li> <li>• "trex": Multi-animal tracking from TRex (2 datasets). The default, "five-locusts", unpacks to one <code>.npz</code> per individual and returns a vector of paths, which <code>read_trex()</code> reads as one recording</li> </ul> <p>Alternatively, provide a URL string (starting with "http://" or "https://") to download a file from a custom location.</p> |
| dataset       | Character string specifying which dataset to download for sources that have multiple options. If NULL (default), the first listed dataset is used. Call <code>get_sample_data(list_datasets = TRUE)</code> to see all available options.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| cache_dir     | Character string specifying the directory where to cache the downloaded files. Defaults to a temporary directory using <code>tempdir()</code> . Set to a permanent location to persist data across R sessions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| quiet         | TRUE/FALSE. TRUE suppresses inform messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| list_datasets | TRUE/FALSE. If TRUE, prints available sources and datasets. Can be called with or without specifying a source.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Details

The function downloads sample data and caches it locally. If the file already exists in the cache directory, it will use the cached version instead of downloading again.

Some sources have multiple datasets available. The first dataset listed for each source is used by default when `dataset = NULL`.

Special handling for TRex datasets: TRex datasets are distributed as zip files containing multiple individual tracking files (one per animal). The function automatically extracts these and returns a vector of paths to the individual files.

The predefined data sources are hosted at:

- <https://gin.swc.ucl.ac.uk/neuroinformatics/movement-sample-data>
- <https://github.com/animovement/movement-data>

## Value

Character string (or vector) with the path(s) to the downloaded file(s). For a dataset distributed as a zip file (the TRex "five-locusts" and trackball datasets), returns a character vector of paths to the files it unpacks to. For all others, returns a single file path. Returns `NULL` invisibly if `list_datasets = TRUE`.

## Examples

```
# See all available sources and datasets
get_sample_data(list_datasets = TRUE)

# See datasets for a specific source
get_sample_data("sleap", list_datasets = TRUE)

# Downloading needs an internet connection
# Get default dataset for SLEAP
path <- get_sample_data("sleap")

# Get a specific SLEAP dataset
path <- get_sample_data("sleap", dataset = "zebras_drone")

# Get TRex data (returns vector of paths to individual files)
paths <- get_sample_data("trex")

# Download from a custom URL
# path <- get_sample_data("https://example.com/data/tracking.csv")
```

---

get\_supported\_sources *List the formats aniread can read and write*

---

## Description

Returns the tracking / event software and file formats that aniread supports, with the function that reads or writes each and the file suffix(es) it handles. This lets downstream packages discover the supported formats programmatically instead of hard-coding the list - mirroring movement's `get_supported_source_software()`.

## Usage

```
get_supported_sources()
```

## Details

Suffixes are returned without a leading dot (e.g. "csv", "h5"), matching the convention used throughout aniread (see the `expected_suffix` argument of the internal file validator). The generic `read_custom()` reader is intentionally omitted because it has no fixed source software or file suffix.

The source names of the "read" rows are exactly those accepted by the `source` argument of `read_dataset()`, and returned by `detect_source()`. Those of the "write" rows are accepted by the `format` argument of `write_dataset()`.

## Value

A **tibble** with one row per supported source and direction, and the columns:

`source` Character. The source software / format name.

`direction` Character. "read" or "write".

`fun` Character. The aniread function that reads or writes it.

`suffix` List column of character vectors - the file suffix(es) the function reads or writes.

`format` List column of character vectors - the export layouts a reader tells apart, as it records them in the `source_format` metadata field. Empty for the "write" rows, and for `aniframe`, whose files keep the `source_format` of the data they hold.

## Examples

```
get_supported_sources()

# The formats aniread can write:
supported <- get_supported_sources()
supported[supported$direction == "write", ]

# The export layouts FreeMoCap has written:
readers <- supported[supported$direction == "read", ]
readers$format[[which(readers$source == "freemocap")]]
```

```
# Which sources read HDF5 (`.h5`) files?
readers$source[vapply(readers$suffix, \(s) "h5" %in% s, logical(1))]
```

---

|               |                                             |
|---------------|---------------------------------------------|
| read_aniframe | <i>Read an aniframe from a Parquet file</i> |
|---------------|---------------------------------------------|

---

### Description

Reads movement data from a Parquet file and returns an aniframe object. Parquet files are required as they preserve the metadata necessary for aniframe objects.

### Usage

```
read_aniframe(path)
```

### Arguments

|      |                         |
|------|-------------------------|
| path | Path to a Parquet file. |
|------|-------------------------|

### Value

The aniframe that was written, with its class: an anipoint, an anievent, or another frame built on them, such as an anijoint. time is as it was written; see "Time" in [read\\_dataset\(\)](#).

### Examples

```
path <- tempfile(fileext = ".parquet")
write_aniframe(anicore::example_anipoint(), path)
read_aniframe(path)
```

---

|               |                           |
|---------------|---------------------------|
| read_animalta | <i>Read AnimalTA data</i> |
|---------------|---------------------------|

---

### Description

Read a data frame from AnimalTA. AnimalTA exports tracking data in image (top-left) coordinates; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin.

AnimalTA writes positions in three layouts, and the reader reads all three. Which one it read is recorded in the source\_format metadata field.

- "fixed": the coordinates file of a video tracked with a fixed number of targets (coordinates/<video>\_Coordinates.csv or corrected\_coordinates/<video>\_Corrected.csv once corrected). One row per frame, with a pair of columns per target, X\_Arena<a>\_Ind<i> and Y\_Arena<a>\_Ind<i>. When a target alone in its arena was tracked with AnimalTA's "Separate head from tail" option, its pair is replaced by X\_Arena<a>\_Ind<i>\_Head, Y...\_Head, X...\_Tail and Y...\_Tail. Once the tracking is corrected, AnimalTA holds them as two targets and names the same columns X\_Arena<a>\_Ind<i>\_part0 and so on for the head and ...\_part1 for the tail. Both become the keypoints head and tail of Ind<i>; AnimalTA writes no centroid for such a target. Every other target has the keypoint centroid, AnimalTA's name for the one position it tracks.
- "variable": the coordinates file of a video tracked with a variable number of targets. One row per target and frame, in the columns Frame, Time, Arena, Ind, X and Y.
- "detailed": a file from the "Detailed data" folder that AnimalTA's "Run analyses" writes, Results/Detailed\_data/<video>/Arena\_<a><target>.csv. It holds one target, one row per frame, and the columns ticked under "Detailed data columns": by default Time, X, Y (X\_Smoothed and Y\_Smoothed when a smoothing filter was applied) and measures derived from them, such as Distance and Speed. Only the time and positions are kept. The target is named from the file name. Several files, one per target, can be read at once by passing their paths together. A target tracked with "Separate head from tail" has two files, Arena\_<a>Ind<i>\_part0.csv for its head and Arena\_<a>Ind<i>\_part1.csv for its tail, which read together as the keypoints head and tail of one individual.

AnimalTA's head and tail are the two ends of the animal's skeleton, kept apart from frame to frame by distance. AnimalTA does not work out which end is the head, so check that head is the head before relying on it.

Positions in the coordinates files are in pixels. Positions in a detailed file are in the unit of the scale set in AnimalTA, and in pixels when none was set. The file does not record which, so unit\_space keeps its default of "px"; if a scale was set, declare its unit with `anicore::set_metadata()`, and give video\_height in that unit too.

AnimalTA writes the time since the start of the video in seconds in its Time column, rounded to 0.01 s in the coordinates files and to 0.001 s in a detailed file. The reader keeps it as time, so unit\_time is "s". A detailed file exported without the Time column but with Frame is timed by its frame number instead, with unit\_time "frame".

AnimalTA does not write the frame rate, but each row is a frame at the rate it tracked at: the coordinates files give its number in Frame, and a detailed file has a row for every frame. When every row agrees on one rate, to within the rounding of Time, it becomes sampling\_rate: the frames between the first and last row over the seconds between them, or the whole number nearest to it when that fits every row as well. Otherwise it is left NA. In a short file the rounding leaves the rate uncertain by about one rounding step over the file's duration.

AnimalTA numbers individuals from 0 within each arena, so an individual is identified by its arena and its name together. The arena is its own identity column, arena, holding AnimalTA's arena number ("0", "1", ...), and individual holds AnimalTA's name for the target: Ind<i>, or the name it was given in AnimalTA, with its case kept. Every layout gives the same values: the fixed layout's columns are named X\_Arena<a>\_Ind<i>, the variable layout's Arena and Ind columns hold the same arena and target, and so does a detailed file's name, Arena\_<a>Ind<i>.csv. A detailed file whose name is not in that form is named after the file, with its arena NA. arena, individual and keypoint are the aniframe's identity keys, so functions that work per individual keep arenas apart. The arena column is there for every file, also when it has a single arena.

Usage

```
read_animalta(  
  path,  
  format = c("auto", "fixed", "variable", "detailed"),  
  video_height = NULL  
)
```

Arguments

|              |                                                                                                                                                                                                                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path         | Path to an AnimalTA file. Several detailed files, one per target, can be given together.                                                                                                                                                                                                                   |
| format       | <b>[Experimental]</b> Which layout the file uses: "auto" (the default) reads it from the header; "fixed", "variable" or "detailed" require that layout and error on anything else. The layout names may change while one convention for readers of a source with several export layouts is settled (#118). |
| video_height | Optional numeric height of the source video frame, in pixels for the coordinates files. AnimalTA does not record this in the export, so when not supplied the maximum observed y is used as a fallback.                                                                                                    |

Value

a movement dataframe. time is AnimalTA's Time (or Frame), with the first frame of the video at 0; see "Time" in `read_dataset()`.

References

- Chiara, V., & Kim, S.-Y. (2023). AnimalTA: A highly flexible and easy-to-use program for tracking and analysing animal movement in different environments. *Methods in Ecology and Evolution*, 14, 1699–1707. doi:10.1111/2041210X.14115.

Examples

```
path <- system.file("extdata", "animalta.csv", package = "aniread")  
read_animalta(path)
```

---

|              |                                             |
|--------------|---------------------------------------------|
| read_anipose | <i>Read Anipose 3D pose estimation data</i> |
|--------------|---------------------------------------------|

---

Description

Reads CSV files output by Anipose's 3D triangulation pipeline and converts them into aniframe format. The function handles the pivot from wide format (one column per keypoint-coordinate combination) to long format with separate rows for each keypoint at each time point.

Usage

```
read_anipose(path, unit_space = "mm")
```

**Arguments**

|            |                                                                                                                                                                                        |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path       | Character string specifying the path to an Anipose CSV file. Typically these are found in the pose-3d directory of your Anipose output.                                                |
| unit_space | Character string specifying the spatial units of the coordinates. This should match the units you used for your calibration board dimensions (e.g., "mm", "cm", "m"). Default is "mm". |

**Value**

An aniframe (tibble) with the following columns:

- time: Frame number, Anipose's fnum, with the first frame of the video at 0; see "Time" in [read\\_dataset\(\)](#)
- keypoint: Name of the tracked body part
- x, y, z: 3D coordinates in the specified units
- confidence: Mean detection confidence across cameras (0-1 scale)

The aniframe includes metadata about the data source, filename, spatial units, and coordinate system.

**Anipose Data Structure**

Anipose outputs CSV files with columns like bodypart\_x, bodypart\_y, bodypart\_z, bodypart\_error, bodypart\_ncams, and bodypart\_score for each tracked keypoint. The coordinates are in whatever units you specified for your calibration board dimensions.

**References**

- Karashchuk, P., Rupp, K. L., Dickinson, E. S., Walling-Bell, S., Sanders, E., Azim, E., Brunton, B. W., & Tuthill, J. C. (2021). Anipose: A toolkit for robust markerless 3D pose estimation. *Cell Reports*, 36(13), 109730. doi:10.1016/j.celrep.2021.109730.

**Examples**

```
path <- system.file("extdata", "anipose.csv", package = "aniread")

# Read anipose data with default millimeter units
read_anipose(path)

# Read with centimeter units
read_anipose(path, unit_space = "cm")
```

read\_bonsai

*Read centroid tracking data from Bonsai***Description**

Read a Bonsai data frame. Bonsai centroid coordinates come from camera/video pipelines that use image (top-left) origin; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin.

Bonsai workflows are user-defined, so what the CSV records about the video depends on the workflow. When it writes the image alongside the centroid, every row carries the image's `Size.Width` and `Size.Height`. When all the images in the file have the same size, that is the frame size: y is reflected around its height, and its width and height are recorded as the `axis_extents` of x and y. Otherwise pass `video_height`, or the largest y is used as the height and no extent is recorded for x.

When the workflow records the blob's `Orientation`, it is kept as `orientation_axis`: the angle of the blob's long axis, in radians from x toward y, in  $(-\pi/2, \pi/2]$ . It is axial — the blob has no front — so it is not declared as the frame's orientation (a yaw), and it is turned with y when y is reflected.

`time` is the seconds elapsed since the first `Timestamp`, so `unit_time` is "s", and the first `Timestamp` itself is kept as `start_datetime`. `sampling_rate` is left NA. A Bonsai `Timestamp` is the time the software received each frame, not when the camera captured it, so the intervals between rows vary from frame to frame and do not state the camera's rate. If you know it, set it with `anicore::set_metadata()`.

**Usage**

```
read_bonsai(path, video_height = NULL)
```

**Arguments**

|                           |                                                                                                                 |
|---------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>path</code>         | Path to a Bonsai data file                                                                                      |
| <code>video_height</code> | Optional numeric height of the source video frame in pixels. Takes precedence over the height the file records. |

**Value**

a movement dataframe. `time` is in seconds from the first row; see "Time" in `read_dataset()`.

**References**

- Lopes, G., Bonacchi, N., Frazão, J., et al. (2015). Bonsai: an event-based framework for processing and controlling data streams. *Frontiers in Neuroinformatics*, 9, 7. doi:10.3389/fninf.2015.00007.

**Examples**

```
path <- system.file("extdata", "bonsai.csv", package = "aniread")
read_bonsai(path)
```

read\_boris

*Read events from a BORIS export*

## Description

Read behavioural events from a **BORIS** export into an `anicores::anievent()`. Two flat-text BORIS exports are supported: **aggregated events** (one row per bout, the default export shape) and **tabular events** (one row per START / STOP / POINT transition; paired into bouts by the reader).

Time units come from the columns BORIS provides. The default `unit_time = "s"` uses Start (s) / Stop (s) and works on any BORIS export. With `unit_time = "frame"` the reader uses the Image index start / Image index stop columns instead, with the first frame of the video at 0 (BORIS counts a video's frames from 0, and the images of an observation of images from 1, which are shifted so the first image is 0); frames stay aligned with rows of a host `anicores::anipoint()`, which keeps event timing robust against effective-FPS drift when the export is paired with movement data. If "frame" is requested but the export carries no image-index columns, the reader falls back to "s" with an informational message. FPS is recorded as `sampling_rate` metadata without rescaling the timestamps; call `anicores::convert_unit_time()` later if you need to convert between frames and seconds.

Channels: each row's channel is the value of BORIS's Behavioral category column when populated, falling back to the literal "behavior" otherwise, including where the aggregated export writes "Not defined" for a behaviour without a category; label is the behaviour name, and type is "state" or "point" mapped from BORIS's Behavior type column. Overlap between bouts of the same channel is permitted on the anievent side; `anicores::validate_anievent()` flags overlapping state bouts with a warning rather than rejecting them. Modifiers travel via the optional modifiers list-column; the multi-column (Modifier #1, Modifier #2, ...) layout, the legacy pipe-separated single-column (a|b|None) layout, and comma-separated values within a single slot are all accepted.

## Usage

```
read_boris(
  path,
  format = c("auto", "aggregated", "tabular"),
  unit_time = c("s", "frame")
)
```

## Arguments

|           |                                                                                                                                                                   |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path      | Path to a BORIS export (.tsv or .csv).                                                                                                                            |
| format    | One of "auto", "aggregated", "tabular". The default "auto" sniffs the format from the first row of the file.                                                      |
| unit_time | One of "s", "frame". Default "s". "frame" uses the BORIS image-index columns; pass it when pairing the anievent with an anipoint to keep frame-aligned semantics. |

**Value**

An `anicore::anievent()` with metadata fields `source`, `source_format` (the format read), `filename`, `unit_time`, and `sampling_rate` populated. `start` and `stop` are BORIS's times in seconds, or its frame numbers with the first frame of the video at 0; see "Time" in `read_dataset()`. The sampling rate is set when every media file in the export has the same FPS; an observation of several media files lists one FPS per file in a single cell, separated by ; (and | between players).

**References**

- Friard, O., & Gamba, M. (2016). BORIS: a free, versatile open-source event-logging software for video/audio coding and live observations. *Methods in Ecology and Evolution*, 7(11), 1325-1330. doi:10.1111/2041210X.12584.

**Examples**

```
path <- system.file("extdata", "boris.csv", package = "aniread")
read_boris(path)
```

---

read\_c3d

---

*Read a C3D motion capture file*


---

**Description**

Reads a C3D file and returns the data as an `aniframe` with associated metadata including source software, units, and sampling rate.

**Usage**

```
read_c3d(path)
```

**Arguments**

`path`                      Path to a .c3d file.

**Details**

C3D numbers frames from 1, and a file's header records the frame of the recording its first frame is, which is not frame 1 when the recording was trimmed. `time` is in seconds from the first frame of the recording, so a file that starts at frame 1 starts at `time = 0`, and one trimmed to start at frame 705 of a 200 Hz recording starts at 3.52 s. A recording longer than 65535 frames records its first frame in the `TRIAL` group's `ACTUAL_START_FIELD` parameter instead, which is used when it is there.

**Value**

An `aniframe` with columns `time`, `keypoint`, `x`, `y`, and `z`. Metadata includes source software, file-name, time/space units, and sampling rate. `time` is in seconds, with the first frame of the recording at 0; see "Time" in `read_dataset()`.

**See Also**

`c3dr::c3d_read()` for lower-level C3D access.

**Examples**

```
# Five frames of a static trial from Vicon Nexus, at 100 Hz
path <- system.file("extdata", "c3d.c3d", package = "aniread")
read_c3d(path)
```

---

|             |                                  |
|-------------|----------------------------------|
| read_custom | <i>Read a custom file format</i> |
|-------------|----------------------------------|

---

**Description**

Reads a file and selects/renames specified columns to create an aniframe.

**Usage**

```
read_custom(
  path,
  cols,
  variables_what = NULL,
  variables_when = NULL,
  variables_where = NULL,
  index = NULL,
  metadata = list()
)
```

**Arguments**

|                 |                                                                                                                                                                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path            | Character string specifying the path to the file to read.                                                                                                                                                                                                                                                                           |
| cols            | A named vector specifying which columns to keep and how to rename them, e.g. <code>c(time = "time", x = "x", y = "y")</code> . Names should be the desired output column names, and values can be either the original column names (as characters) or column positions (as integers, e.g. <code>c(time = 1, x = 2, y = 3)</code> ). |
| variables_what  | Character vector of identity columns that together define a unique entity. Uses aniframe defaults if NULL.                                                                                                                                                                                                                          |
| variables_when  | Character vector of temporal columns that together define a unique timepoint. Uses aniframe defaults if NULL.                                                                                                                                                                                                                       |
| variables_where | Character vector of spatial columns that together define position. If NULL, detected from data.                                                                                                                                                                                                                                     |
| index           | Length-one character vector naming the column the frame is indexed by – the position of each row within its temporal context. If NULL (the default), "time". It is never a grouping variable, and is declared separately from <code>variables_when</code> .                                                                         |
| metadata        | A list of metadata to attach to the resulting aniframe. Default is an empty list.                                                                                                                                                                                                                                                   |

**Value**

An aniframe with the selected and renamed columns, and attached metadata. `time` is the column as it is in the file; see "Time" in `read_dataset()` for the convention the other readers follow.

**Examples**

```
# A small file with columns named the tracker's way
path <- tempfile(fileext = ".csv")
write.csv(
  data.frame(
    frame = c(1, 2, 3, 1, 2),
    pos_x = c(0.1, 0.4, 0.9, 2.0, 2.2),
    pos_y = c(1.0, 1.2, 1.1, 0.5, 0.7),
    track_id = c(1, 1, 1, 2, 2),
    trial_num = 1
  ),
  path,
  row.names = FALSE
)

# Simple case: rename columns to standard names, use defaults
read_custom(path, cols = c(time = "frame", x = "pos_x", y = "pos_y"))

# With identity variable
read_custom(
  path,
  cols = c(track = "track_id", time = "frame", x = "pos_x", y = "pos_y"),
  variables_what = "track"
)

# Full specification with trials
read_custom(
  path,
  cols = c(
    id = "track_id",
    trial = "trial_num",
    frame = "frame",
    x = "pos_x",
    y = "pos_y"
  ),
  variables_what = "id",
  variables_when = "trial",
  index = "frame"
)

# Using column positions
read_custom(path, cols = c(time = 1, x = 2, y = 3))
```

## Description

One entry point for every format aniread supports. By default the source software is worked out from the file itself, so you do not have to know which reader a file needs before opening it.

## Usage

```
read_dataset(paths, source = "auto", ...)
```

## Arguments

|        |                                                                                                                                                                                                                                                    |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| paths  | Path to the file to read. A few readers take more than one path - <a href="#">read_trackball()</a> takes one per sensor - in which case pass them all; detection inspects the first.                                                               |
| source | Which source software wrote the file. "auto" (the default) detects it with <a href="#">detect_source()</a> . Otherwise one of the names in <a href="#">get_supported_sources()</a> .                                                               |
| ...    | Passed on to the reader for source. This is how arguments that only some readers take are supplied, e.g. <code>sampling_rate</code> for <a href="#">read_trackball()</a> or <code>path_probabilities</code> for <a href="#">read_idtracker()</a> . |

## Details

`read_dataset()` is a dispatcher, not a new reader: it works out which reader to call and calls it. The object you get back is exactly what the underlying reader returns - an [anipoint](#) for tracking data, or an [anievent](#) for behavioural events from [read\\_boris\(\)](#).

DeepLabCut and LightningPose CSV exports are structurally identical, so a file in that format cannot be attributed to one or the other. Such a file is read with [read\\_deeplabcut\(\)](#) - the parse is the same either way - and its source metadata is set to "deeplabcut/lightningpose" to record that the distinction is undetermined. Pass source explicitly to override this.

Every reader records the export it read in the `source_format` metadata field, so a file in a layout a reader does not expect can be traced back to it; [get\\_supported\\_sources\(\)](#) lists the values each reader records. The version of the software that wrote the file goes in `source_version` only where the file states it, and is NA otherwise.

## Value

An [anipoint](#) or [anievent](#), depending on the reader. time (or an event's start and stop) follows the convention in the Time section.

## Time

Every reader gives time as the time elapsed since the first frame of the video, so the first frame of the video is at time = 0, whatever the unit ([#150](#)):

- In frames, time is the frame number counted from 0: the first frame of the video is 0, the next 1, and so on. Converting to seconds is then a division by the frame rate, with no offset.
- In seconds, the first frame of the video is at 0 s.
- Offsets are kept. A file whose first row is frame 500 of the video has time = 500 there, not 0, so it stays aligned with the video.

Sources that count frames from 0 are read as they are. Sources that count them from 1 are shifted by one. Where a source gives times in seconds, they are kept. The table lists each source's own convention, what time was before aniread 0.8.0 and what it is now. Sources not marked as changed read as before.

| Source (reader)                                      | The source's own time                                                                            |
|------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <code>aniframe (read_aniframe())</code>              | As written by <code>write_aniframe()</code>                                                      |
| <code>AnimalTA (read_animalta())</code>              | Frame counts the video's frames from 0, keeping a cropped start; Time is the frame number        |
| <code>Anipose (read_anipose())</code>                | fnum counts frames from 0                                                                        |
| <code>Bonsai (read_bonsai())</code>                  | A clock time per row, set by the workflow; no frame number                                       |
| <code>BORIS (read_boris())</code>                    | Start (s)/Stop (s) are media time in s. The image index counts a video's frame number            |
| <code>C3D (read_c3d())</code>                        | Frames count from 1; the header records the recording frame the file starts at                   |
| <code>DeepLabCut (read_deeplabcut())</code>          | The row index counts frames from 0                                                               |
| <code>FastTrack (read_fasttrack())</code>            | imageNumber is the video's frame, from 0                                                         |
| <code>FicTrac (read_fictrac())</code>                | A clock time per row, in ms                                                                      |
| <code>FreeMoCap (read_freemocap())</code>            | frame counts from 0, or the row is the frame; v1 also writes timestamps                          |
| <code>idtracker.ai .h5 (read_idtracker())</code>     | Rows are the video's frames, from 0                                                              |
| <code>idtracker.ai CSV, with a time column</code>    | time (or seconds) is the frame over the frame rate, in s                                         |
| <code>idtracker.ai CSV, without a time column</code> | Rows are the video's frames, from 0                                                              |
| <code>idtracker.ai tidy CSV and Parquet</code>       | frame counts from 0; time is the frame over the frame rate, or over 1 when the rate is not known |
| <code>Lightning Pose (read_lightningpose())</code>   | The row index counts frames from 0                                                               |
| <code>movement (read_movement())</code>              | The time coordinate: frames from 0, or the frame over the frame rate in s                        |
| <code>OCTRON (read_octron())</code>                  | frame_idx is the video's frame, from 0                                                           |
| <code>SLEAP .h5 (read_sleap())</code>                | The frame axis starts at frame_idx 0                                                             |
| <code>SLEAP CSV</code>                               | frame_idx counts from 0                                                                          |
| <code>Trackball (read_trackball())</code>            | Sensor samples with a clock or counter; no video                                                 |
| <code>TrackMate (read_trackmate())</code>            | FRAME counts from 0; POSITION_T is the frame times the frame interval                            |
| <code>TRex (read_trex())</code>                      | frame counts from 0; time is in s, 0 at frame 0                                                  |
| <code>read_custom()</code>                           | Whatever the file holds                                                                          |

Where a source keeps only clock times (Bonsai, FicTrac, the FreeMoCap v1 timestamps), nothing in the file marks the first frame of the video, so the first row is taken as 0. That is the first frame the tool logged: FicTrac writes no row for a frame it could not match, so if it lost its first frames, 0 is the first one it matched. A SLEAP .h5 written without its leading untracked frames (sleap-io's `all_frames = FALSE`) does not record where it starts, so its first row reads as frame 0.

### See Also

`detect_source()` to detect the format without reading, `get_supported_sources()` for what is supported, and the individual `read_*`() functions for format-specific arguments.

### Examples

```
# Let aniread work out the format
path <- system.file("extdata", "sleap.analysis.csv", package = "aniread")
read_dataset(path)

# Name it explicitly
```

```

read_dataset(path, source = "sleap")

# Reader-specific arguments pass straight through
paths <- c(
  system.file("extdata", "trackball_sensor_1.csv", package = "aniread"),
  system.file("extdata", "trackball_sensor_2.csv", package = "aniread")
)
read_dataset(
  paths,
  setup = "of_free",
  sampling_rate = 60,
  col_time = "datetime"
)

```

---

|                 |                             |
|-----------------|-----------------------------|
| read_deeplabcut | <i>Read DeepLabCut data</i> |
|-----------------|-----------------------------|

---

## Description

Read files from DeepLabCut (DLC) in either csv or h5 format. DeepLabCut stores predictions in image (top-left) coordinates; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin. DLC's csv/h5 exports do not contain the source video resolution (it lives in the project's config.yaml), so pass video\_height to get an accurate flip; otherwise max(y) is used as a fallback.

## Usage

```
read_deeplabcut(path, video_height = NULL)
```

## Arguments

|              |                                                                                     |
|--------------|-------------------------------------------------------------------------------------|
| path         | Path to a DeepLabCut data file                                                      |
| video_height | Optional numeric height of the source video frame in pixels. Not used for 3D files. |

## Value

an aniframe. time is DeepLabCut's frame index, with the first frame of the video at 0; see "Time" in [read\\_dataset\(\)](#).

## Layouts read

The file is read from its column header, the scorer, individuals (multi-animal projects only), bodyparts and coords levels, so the same data reads the same from a csv and an h5. An h5 is read whichever way pandas stored it: DeepLabCut writes the "table" format, while a file saved again with pandas' defaults, as movement's to\_dlc\_file() does, is in the "fixed" format. Predictions are read from the HDF key df\_with\_missing, and the stitched tracklets of a multi-animal project (\*\_el.h5) from the key tracks.

Of the coords, x, y, z and likelihood are read, and likelihood becomes confidence. Any other coords, such as the variances the Ensemble Kalman Smoother adds, are not read.

A multi-animal project tracks the bodyparts it does not assign to an animal (its uniquebodyparts) under the pseudo-individual single. They are kept as the keypoints of an individual called "single".

The layout read is recorded in the source\_format metadata field: the file type, single- or multi-animal, and \_3d for triangulated positions, such as "csv\_single\_animal" or "h5\_multi\_animal".

### 3D files

DeepLabCut's triangulation writes x, y and z and no likelihood, so confidence is NA. Triangulated positions are in the units and frame of the stereo calibration rather than in image pixels, so they are returned as stored: y is not reflected and video\_height is not used. The aniframe is 3D from its z column.

### References

- Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., & Bethge, M. (2018). DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21, 1281-1289. doi:10.1038/s415930180209y.
- Nath, T., Mathis, A., Chen, A. C., Patel, A., Bethge, M., & Mathis, M. W. (2019). Using DeepLabCut for 3D markerless pose estimation across species and behaviors. *Nature Protocols*, 14, 2152-2176. doi:10.1038/s4159601901760.

### Examples

```
path <- system.file("extdata", "deeplabcut.csv", package = "aniread")
read_deeplabcut(path)
```

---

|                |                                     |
|----------------|-------------------------------------|
| read_fasttrack | <i>Read FastTrack tracking data</i> |
|----------------|-------------------------------------|

---

### Description

Reads a FastTrack tracking result file (.txt format) and returns an aniframe with keypoints for head, body, and tail positions. FastTrack stores positions in image (top-left) coordinates; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin. The tracking file does not record the source video resolution, so pass video\_height to get an accurate flip — otherwise max(y) is used as a fallback.

### Usage

```
read_fasttrack(path, video_height = NULL)
```

### Arguments

|              |                                                              |
|--------------|--------------------------------------------------------------|
| path         | Path to a FastTrack tracking.txt file.                       |
| video_height | Optional numeric height of the source video frame in pixels. |

**Value**

An aniframe. `time` is FastTrack's `imageNumber`, with the first frame of the video at 0; see "Time" in `read_dataset()`.

**References**

- Gallois, B., & Candelier, R. (2021). FastTrack: An open-source software for tracking varying numbers of deformable objects. *PLOS Computational Biology*, 17(2), e1008697. doi:10.1371/journal.pcbi.1008697.

**Examples**

```
path <- system.file("extdata", "fasttrack.txt", package = "aniread")
read_fasttrack(path)
```

---

|              |                                    |
|--------------|------------------------------------|
| read_fictrac | <i>Read projected FicTrac data</i> |
|--------------|------------------------------------|

---

**Description**

This helper loads a FicTrac \*.dat file, keeps the timestamp, the 2-D fictive path and the animal's heading, converts the timestamps to seconds, and returns the result as an anipoint. FicTrac's "integrated animal heading" (the direction the animal faces) becomes yaw and is declared as the frame's orientation, in radians from x toward y. FicTrac's movement direction is the direction of travel, derivable from the path, and is not kept. If the physical ball radius is supplied, the positions are scaled accordingly and the spatial unit metadata is set.

**Usage**

```
read_fictrac(path, ball_radius = NULL, unit_ball_radius = "cm")
```

**Arguments**

|                               |                                                                                                                                    |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>path</code>             | Character. Path to the FicTrac *.dat file.                                                                                         |
| <code>ball_radius</code>      | Numeric (optional). Physical radius of the tracking ball. When supplied the x and y coordinates are multiplied by this value.      |
| <code>unit_ball_radius</code> | Character. Unit of <code>ball_radius</code> (e.g., "cm" or "mm"). Defaults to "cm". Ignored when <code>ball_radius</code> is NULL. |

**Details**

FicTrac 2 has written three layouts, told apart by their number of columns. Columns 1 to 21 are the same in all of them; the rest hold timestamps:

- **23 columns** (FicTrac 2.0 to 2.02): 22 is the timestamp, 23 the sequence counter. `time` is taken from the timestamp, which is the position in the video file (ms) or the frame capture time (ms).

- **24 columns** (untagged versions from July 2019, before 2.03): 22 is the frame capture time in ms since midnight, 23 the sequence counter, 24 the time since the last frame (ms). time is taken from column 22.
- **25 columns** (FicTrac 2.03 onward): 22 is the timestamp, 23 the sequence counter, 24 the time since the last frame, 25 the "alt. timestamp", the frame capture time in ms since midnight. time is taken from column 25.

Which layout was read is recorded in the `source_format` metadata field, as "23col", "24col" or "25col".

time counts in seconds from the first row. A time in ms since midnight starts again from zero at midnight, so where it drops by more than twelve hours from one row to the next, the reader takes it as having passed midnight and adds a day from there on. A recording that crosses midnight therefore keeps running forward rather than jumping back.

### Value

An anipoint with columns `time`, `x`, `y` and `yaw`. Metadata includes the source ("fictrac"), original filename, sampling rate, time unit ("s"), space unit (either "none" or the value of `unit_ball_radius`), and a Cartesian 2-D coordinate system. time is in seconds from the first row; see "Time" in `read_dataset()`.

### References

- Moore, R. J. D., Taylor, G. J., Paulk, A. C., Pearson, T., van Swinderen, B., & Srinivasan, M. V. (2014). FicTrac: a visual method for tracking spherical motion and generating fictive animal paths. *Journal of Neuroscience Methods*, 225, 106-119. doi:10.1016/j.jneumeth.2014.01.010.

### Examples

```
path <- system.file("extdata", "fictrac.dat", package = "aniread")
traj <- read_fictrac(path, ball_radius = 0.5, unit_ball_radius = "cm")
head(traj)
```

---

read\_freemocap

*Read FreeMoCap motion capture data*

---

### Description

Reads the layouts FreeMoCap v1 and v2 write, dispatching on the column names rather than the file name:

- `<recording>_by_frame.csv`: the v1 tidy export. FreeMoCap added a `reprojection_error` column at v1.7.4, so this exists in an 8- and a 9-column form; both are read.
- `<recording>_by_trajectory.csv`: v1's one column triple per tracked point, with the camera timestamps alongside.
- `output_data/mediapipe_*_3d_xyz.csv`: v1's per-model wide files, one model each and no timestamps.

- output\_data/freemocap\_data\_by\_frame.csv, and the .parquet beside it: the v2 tidy export, one row per keypoint per trajectory per frame (see trajectory). Reading the Parquet file needs the arrow package.
- output\_data/<tracker>\_<aspect>\_<trajectory>.csv: v2's per-trajectory files, such as mediapipe\_body\_3d\_xyz.csv. They keep v1's names but are long (frame, keypoint, x, y, z), so they are told from v1's wide files by their columns. Their model is read from the file name.

Which layout was read is recorded in the source\_format metadata field. Point names are parsed the way FreeMoCap's own data saver parses them, so the same recording gives the same model and keypoint values whichever layout it is read from.

## Usage

```
read_freemocap(
  path,
  format = c("auto", "by_frame", "by_trajectory", "wide", "v2_by_frame", "v2_trajectory"),
  trajectory = c("3d_xyz", "rigid_3d_xyz")
)
```

## Arguments

|            |                                                                                                                                                                                                                                                                                                                                                                               |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path       | Path to a FreeMoCap CSV, or a v2 Parquet file.                                                                                                                                                                                                                                                                                                                                |
| format     | <b>[Experimental]</b> Export layout. "auto" (default) reads it from the column names; naming one requires that layout and errors on anything else. The layout names may change while one convention for readers of a source with several export layouts is settled ( <a href="#">#118</a> ).                                                                                  |
| trajectory | Which positions to read from the v2 tidy export: "3d_xyz" (default), the triangulated, filtered keypoints, or "rigid_3d_xyz", the same keypoints with bone lengths held constant. FreeMoCap only computes rigid_3d_xyz for aspects with a joint hierarchy (the body), so other aspects keep 3d_xyz and a message says which. Ignored, with a warning, for every other layout. |

## Details

confidence comes from reprojection\_error, which the 9-column by\_frame export and the v2 tidy export carry; every other layout gives all-NA confidence. The two run in opposite directions: a reprojection error is a distance in pixels, so zero is perfect and larger is worse, whereas every other reader in aniread fills confidence from a likelihood or a probability where larger is better. Storing the error unchanged would make aniprocess::mask\_na\_across(method = "confidence") mask the best points, so it is mapped through

$$confidence = 1/(1 + error)$$

which is monotone decreasing onto (0,1]: a zero error gives 1. The mapping is invertible, so the original error is recoverable as 1 / confidence - 1.

### FreeMoCap v2:

v2 writes its output through skellyforge. Its tidy export names each model `<tracker>.<aspect>` (`mediapipe.body`, `rtmpose.left_hand`); these are read as `<tracker>_<aspect>` (`mediapipe_body`, `rtmpose_left_hand`), the naming of v1's model column and of v2's own per-trajectory file names. The hands keep separate models, since v2's hand keypoint names do not say which hand they belong to.

Each keypoint appears once per trajectory type, so only one set of positions is read, chosen by trajectory. The centres of mass (`total_body_center_of_mass` and `segment_center_of_mass`) are added as keypoints of a `<tracker>_com` model, as v1 kept them in `mediapipe_com`, with the keypoint names v2 gives them. (Only the body has centres of mass in FreeMoCap's models; another aspect's would go to `<tracker>_<aspect>_com`, so that segment names cannot collide.) Any other trajectory type is not read.

v2 writes no timestamps, so time is the frame number and `sampling_rate` is left unset. FreeMoCap's main pipeline does not attach reprojection errors to the skeleton it saves, so confidence is usually all NA too.

`detect_source()` recognises the v2 tidy CSV. It does not claim the per-trajectory files, whose five columns any tidy export could have, or the Parquet file; read those with `read_freemocap()` directly.

## Value

An aniframe with time, model, keypoint, confidence and x/y/z in millimetres on a 3D cartesian coordinate system. time is seconds elapsed from `start_datetime` where the layout carries timestamps, and frames where it does not. Either way the first frame is at 0; see "Time" in `read_dataset()`.

## References

- Matthis, J. S., Queen, P., Cherian, A., Wirth, T., Idehen, E., et al. FreeMoCap: a free, open-source markerless motion capture system (software). doi:[10.5281/zenodo.7233713](https://doi.org/10.5281/zenodo.7233713).

## Examples

```
path <- system.file("extdata", "freemocap.csv", package = "aniread")
read_freemocap(path)

# The same recording in its by_trajectory form
path <- system.file(
  "extdata",
  "freemocap_by_trajectory.csv",
  package = "aniread"
)
read_freemocap(path)

# FreeMoCap v2's tidy export, reading the rigid-body positions
path <- system.file("extdata", "freemocap_v2.csv", package = "aniread")
read_freemocap(path, trajectory = "rigid_3d_xyz")
```

---

|                |                               |
|----------------|-------------------------------|
| read_idtracker | <i>Read idtracker.ai data</i> |
|----------------|-------------------------------|

---

## Description

idtracker.ai stores trajectories in image (top-left) coordinates; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin. For h5 files the frame height is read from the file, as described below. CSV exports do not include the frame height, so pass video\_height explicitly to get an accurate flip (otherwise max(y) is used as a fallback).

## Usage

```
read_idtracker(
  path,
  path_probabilities = NULL,
  video_height = NULL,
  format = c("auto", "h5", "csv", "csv_tidy", "parquet")
)
```

## Arguments

|                    |                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path               | Path to an idtracker.ai data frame                                                                                                                                                                                                                                                                              |
| path_probabilities | Path to a csv file with probabilities. Only needed if you are reading csv files as they are included in h5 files.                                                                                                                                                                                               |
| video_height       | Optional numeric height of the source video frame in pixels. Overrides the value read from the h5 or Parquet file when both are available.                                                                                                                                                                      |
| format             | Which idtracker.ai export path is: "h5", the CSV export ("csv", trajectories.csv), the tidy CSV export ("csv_tidy", trajectories_tidy.csv) or "parquet". The default, "auto", tells them apart by the suffix and, for a CSV, by its header. Which one was read is recorded in the source_format metadata field. |

## Details

An h5 file also records, as attributes of its root, how the recording was tracked, and the reader keeps what has a place in the metadata:

- version, the idtracker.ai version that tracked it, becomes the source\_version metadata field.
- frames\_per\_second, the frame rate of the video, becomes the sampling\_rate metadata field. time stays the frame number, counted from 0 as idtracker.ai counts frames, so unit\_time is still "frame"; the frame rate is what converts it to seconds.
- height, the height of the video frame, is what y is reflected around, unless video\_height is given. A height dataset is used when there is no such attribute.
- width, the width of the video frame, written by newer releases, becomes the x extent in the axis\_extents metadata field.

- `identities_labels`, the names of the identities, which can be changed in `idtracker.ai`'s validator, name the individuals. `idtracker.ai` writes "1", "2", ... when none were set, which are also the names used when the file has no usable labels (one distinct, non-empty label per individual).

`source_version` and `sampling_rate` stay NA when the file does not record them, and no x extent is recorded without a width.

The CSV export keeps these in a separate `attributes.json` rather than in `trajectories.csv`, and the reader does not read it, so a frame read from the CSV has no `source_version`; set it with `anicode::set_metadata()`. Its time column (seconds, or time in newer releases) is the time in seconds, so `unit_time` is "s", where a frame read from the h5 has "frame". The time column still states the frame rate: `idtracker.ai` writes one row per frame, with the time as the row number divided by the frame rate, rounded to 1 ms. The reader takes the rate from the rows over the time they span, or the whole number nearest to it when that fits every row as well, and sets it as `sampling_rate`. It is left NA when the times are not evenly spaced.

When `idtracker.ai` could not read the video's frame rate, it writes the CSV export without a time column, and `frames_per_second` as null in `attributes.json`. `time` is then the frame number, the row counted from 0 as the h5 reader counts frames, `unit_time` is "frame", and `sampling_rate` is NA. Set the rate with `anicode::set_metadata()` if you know it.

## Value

a movement dataframe. The first frame of the video is at `time = 0`, in frames or in seconds; see "Time" in `read_dataset()`.

## Tidy CSV and Parquet exports

`idtracker.ai` 6.0.13 added a Parquet export, `trajectories.parquet`, and 6.0.14 a tidy CSV export, `trajectories_tidy.csv`. Both hold one row per frame and individual, with the columns `frame`, `time`, `individual`, `x`, `y` and `probability`, and are written only when asked for in `TRAJECTORIES_FORMATS` (or by `idtrackerai_format --formats csv_tidy parquet`).

- The Parquet file stores the attributes the h5 keeps (`version`, `frames_per_second`, `height`, `width`, `identities_labels`, ...) as JSON in its own metadata, and the reader uses them as it does the h5's.
- The tidy CSV keeps them in `attributes_tidy.json` beside it, which the reader reads when it is there, since `idtracker.ai` writes the two together as one export.

`frame` and `individual` count from 0. Individuals are numbered from 1, as the h5 reader numbers them, and named by `identities_labels` where those are usable. `probability` becomes `confidence`. When the frame rate is known, `time` is the file's time in seconds, which is the frame over the frame rate, as the CSV export times its rows, with `unit_time` "s" and the frame rate as `sampling_rate`. When `idtracker.ai` could not read the frame rate it still writes a `time` column, but as the frame over 1, so `time` is then the frame, counted from 0, with `unit_time` "frame" and `sampling_rate` NA. Without `attributes_tidy.json`, a tidy CSV whose `time` equals its `frame` in every row is read as one without a frame rate, and otherwise the rate is taken from the two columns, as for the CSV export.

## References

- Romero-Ferrero, F., Bergomi, M. G., Hinz, R. C., Heras, F. J. H., & de Polavieja, G. G. (2019). idtracker.ai: tracking all individuals in small or large collectives of unmarked animals. *Nature Methods*, 16, 179-182. doi:[10.1038/s4159201802955](https://doi.org/10.1038/s4159201802955).
- Torrents, J., Costa, T., & de Polavieja, G. G. (2025). New idtracker.ai: rethinking multi-animal tracking as a representation learning problem to increase accuracy and reduce tracking times. *eLife* (reviewed preprint). doi:[10.7554/eLife.107602](https://doi.org/10.7554/eLife.107602).

## Examples

```
path <- system.file("extdata", "idtracker.csv", package = "aniread")
read_idtracker(path)
```

---

|                    |                                |
|--------------------|--------------------------------|
| read_lightningpose | <i>Read LightningPose data</i> |
|--------------------|--------------------------------|

---

## Description

Read csv files from LightningPose (LP). Like DeepLabCut, the source is in image (top-left) coordinates and the reader reflects y to bottom\_left.

## Usage

```
read_lightningpose(path, video_height = NULL)
```

## Arguments

|              |                                                                                                      |
|--------------|------------------------------------------------------------------------------------------------------|
| path         | Path to a LightningPose data file                                                                    |
| video_height | Optional numeric height of the source video frame in pixels. Falls back to max(y) when not supplied. |

## Details

Lightning Pose writes DeepLabCut's csv layout, so the file is read by [read\\_deeplabcut\(\)](#). That includes the output of its Ensemble Kalman Smoother (EKS, scorer ensemble-kalman\_tracker), which has nine coords per keypoint: x, y and likelihood are read, and the ensemble medians and variances and the posterior variances (x\_ens\_median, y\_ens\_median, x\_ens\_var, y\_ens\_var, x\_posterior\_var, y\_posterior\_var) are not. Its multi-camera 3D output, multicam\_3d\_results.csv, reads as 3D with x, y and z, without confidence and without reflecting y; see [read\\_deeplabcut\(\)](#).

## Value

an aniframe. time is Lightning Pose's frame index, with the first frame of the video at 0; see "Time" in [read\\_dataset\(\)](#).

References

- Biderman, D., Whiteway, M. R., Hurwitz, C., et al. (2024). Lightning Pose: improved animal pose estimation via semi-supervised learning, Bayesian ensembling and cloud-native open-source tools. *Nature Methods*, 21, 1316-1328. doi:10.1038/s41592024023191.
- Aharon, L., Whiteway, M. R., et al. (2026). Lightning Pose 3D: an uncertainty-aware framework for data-efficient multi-view animal pose estimation. *bioRxiv*. doi:10.64898/2026.04.20.719731.

Examples

```
path <- system.file("extdata", "lightningpose.csv", package = "aniread")
read_lightningpose(path)
```

---

|               |                                                            |
|---------------|------------------------------------------------------------|
| read_movement | <i>Read data exported from the movement Python package</i> |
|---------------|------------------------------------------------------------|

---

Description

Imports pose estimation data from netCDF/HDF5 files created by the **movement** Python package.

Usage

```
read_movement(path, video_height = NULL)
```

Arguments

|              |                                                                                                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path         | Path to an HDF5 file (.nc or .h5) exported from movement.                                                                                                                                                |
| video_height | Optional numeric height of the source video frame in pixels. movement does not currently store this in the netCDF attributes, so without it max(y) is used as a fallback when reflecting to bottom_left. |

Details

The movement package stores pose estimation data in a specific netCDF/HDF5 structure with datasets for individuals, keypoints, position coordinates, confidence scores, and time. This function reads that structure and reshapes it into a tidy aniframe format. The underlying tracking software outputs image (top-left) coordinates, so the reader reflects y to bottom\_left before returning.

Since version 0.17.0, movement names the dimensions individual and keypoint; files saved by earlier versions name them individuals and keypoints. Both read the same way.

The axes are taken from the file's space coordinate, so a 2D dataset gives x and y and a 3D one x, y and z. The confidence variable becomes the confidence column. movement stores it either for each point, with dimensions (time, keypoint, individual), or for each individual, with dimensions (time, individual); a per-individual score is repeated for every keypoint of that individual.

Only poses datasets are read. A bounding boxes dataset (ds\_type "bboxes") or a multi-view dataset, which has a further view dimension, is an error.

The file's root attributes describe the recording, and the reader keeps what has a place in the meta-data:

- `source_software` becomes `source`, and the name of `source_file` becomes `filename`. A file without `source_file` (a dataset movement built from arrays rather than loaded from a file) takes the name of the file read. `source_format` is "netcdf", the format movement saves to, whichever suffix the file has.
- `time_unit` becomes `unit_time`. movement writes "seconds" when it was given the frame rate and "frames" when it was not; these become "s" and "frame".
- `fps` becomes `sampling_rate`. movement writes it only when it was given one, so a file whose time is in frames leaves `sampling_rate` NA. Set it with `anicores::set_metadata()` if you know it.

### Value

An `aniframe`. `time` is movement's time coordinate, with the first frame of the video at 0; see "Time" in `read_dataset()`.

### References

- Sirmipilatze, N., Lo, C. H., Miñano, S., Peri, B. D., Sharma, D., Porta, L., Varela, I., & Tyson, A. L. movement: Python tools for analysing body movements across space and time (software). [doi:10.5281/zenodo.12755724](https://doi.org/10.5281/zenodo.12755724).

### Examples

```
# Two mice tracked with SLEAP, saved by movement with an fps of 50
path <- system.file("extdata", "movement.nc", package = "aniread")
read_movement(path)
```

---

|             |                                      |
|-------------|--------------------------------------|
| read_octron | <i>Read Octron Segmentation Data</i> |
|-------------|--------------------------------------|

---

### Description

Reads CSV files exported from Octron video segmentation software. The function parses the meta-data header and returns tracking data as an `aniframe` with centroid positions, bounding box corners, and shape descriptors. Octron stores positions in image (top-left) coordinates; the reader reflects y so the returned `aniframe` is in the conventional bottom-left origin. The frame height is read from the CSV header (`video_height:`) by default.

### Usage

```
read_octron(
  path,
  keep_bbox = FALSE,
  video_height = NULL,
  method = c("weighted", "largest", "segments"),
  properties = "all"
)
```

**Arguments**

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path         | Path to the Octron CSV file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| keep_bbox    | Keep bounding box coordinates? Default FALSE.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| video_height | Optional numeric height of the source video frame in pixels. Overrides the value parsed from the CSV header when both are available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| method       | <p>Strategy for resolving frames where Octron emitted multiple mask segments per track. One of:</p> <ul style="list-style-type: none"> <li>• "weighted" (default): area-weighted mean across all segments per row. area becomes the sum of segment areas; orientation falls back to "largest" (circular quantity, weighted mean is undefined).</li> <li>• "largest": take values from the single largest segment per row.</li> <li>• "segments": expand each multi-segment row into one row per segment, adding a segment identity variable. Segments are not matched across frames, so filtering on segment is generally not meaningful.</li> </ul> <p>When the source CSV contains no tuple-valued rows, all three methods produce identical numeric output.</p>                                                                                                                                                           |
| properties   | <p>Which scikit-image / Octron region-property columns to read. Newer Octron exports include dozens of per-segment shape, intensity and moment descriptors that can dominate read time on tuple-heavy files. One of:</p> <ul style="list-style-type: none"> <li>• "all" (default): read every property column found in the file. Backwards-compatible with prior read_octron() behaviour.</li> <li>• a character vector of column names, e.g. c("area", "orientation"): read only the listed properties. Unknown names are warned about and ignored.</li> <li>• NULL or character(0): skip all property columns; the result contains only id columns and the centroid (and bbox when keep_bbox = TRUE).</li> </ul> <p>When method = "weighted" and area exists in the file but is absent from this argument, it is added automatically (the area-weighted mean has no meaning otherwise) and an info message is emitted.</p> |

**Details**

Newer Octron exports ( $\geq$  the multi-blob handling in [OCTRON-GUI #63](#)) may emit per-segment columns as tuple-strings, e.g. "(120.5, 85.3)", when YOLO detects multiple disconnected mask segments belonging to the same track in a single frame. The method argument controls how those rows are reduced to scalar values, or whether they are expanded into one row per segment.

**Value**

An anipoint. time is OCTRON's frame\_idx, with the first frame of the video at 0; see "Time" in [read\\_dataset\(\)](#).

**Orientation**

Octron's orientation is scikit-image's region property: the angle of the mask's long axis, axial (the mask has no front), measured from the image's row axis. It is kept as Octron wrote it, in image coordinates: it is not reflected with y and not declared as the frame's orientation.

References

- Jacobsen, R. I., et al. (2025). OCTRON: a general purpose segmentation and tracking pipeline for behavioral experiments. *bioRxiv*. doi:10.64898/2025.12.20.695663.

Examples

```
path <- system.file("extdata", "octron.csv", package = "aniread")
read_octron(path)
```

---

|            |                        |
|------------|------------------------|
| read_sleap | <i>Read SLEAP data</i> |
|------------|------------------------|

---

Description

Reads either of SLEAP’s analysis exports: the HDF5 file, or the CSV with columns track, frame\_idx, instance.score and a .x/.y/.score triple per node.

Usage

```
read_sleap(path, video_height = NULL)
```

Arguments

|              |                                                              |
|--------------|--------------------------------------------------------------|
| path         | A SLEAP analysis file, either HDF5 (.h5) or CSV.             |
| video_height | Optional numeric height of the source video frame in pixels. |

Details

SLEAP stores predictions in image (top-left) coordinates; the reader reflects y so the returned ani-frame is in the conventional bottom\_left origin. Neither export includes the source video resolution, so pass video\_height to get an accurate flip — otherwise max(y) is used as a fallback.

individual is the track name SLEAP recorded, from either export. A recording with no tracks - a single unnamed instance, or predictions that were never tracked - has no names to use, and falls back to individual1, individual2, and so on. In the CSV, where such rows have an empty track, they are numbered in the order they appear within each frame. sleap-io (which writes SLEAP’s exports from SLEAP 1.6.3 on) gives the instances of an untracked recording the names track\_0, track\_1, ... in the .h5, and those are kept like any other track name.

Both the files SLEAP wrote itself and those written by sleap-io are read. sleap-io can store the .h5 arrays in another axis order (its "standard" preset, or custom axes) and records the order in each dataset’s dims attribute, which the reader follows; a file without it has SLEAP’s original layout. The frame axis of a sleap-io .h5 runs to the end of the video, so frames after the last detection come back as NA rows, as undetected frames always have. Very old .h5 files have no point\_scores, and give NA confidence.

The .h5 also carries the skeleton the recording was tracked with and a record of the run, and both are kept:

- The skeleton is attached as the frame's keypoint structure, read as `read_structure_sleap()` reads it: the nodes become points and the body edges segments. A file written without edges gives points alone.
- The SLEAP version that ran the tracking, from the file's provenance record, becomes the `source_version` metadata field. When the record names none, a file written by sleap-io gives the sleap-io version that wrote it instead, as "sleap-io 0.9.2" for example. It stays NA when the file records neither.

The CSV carries neither, so a frame read from it has no structure and no `source_version`. Attach one with `read_structure()` and `anicore::set_structure()`.

### Value

a movement dataframe. `time` is SLEAP's frame index, so the first frame of the video is at `time = 0`; see "Time" in `read_dataset()`.

### References

- Pereira, T. D., Tabris, N., Matsliah, A., et al. (2022). SLEAP: A deep learning system for multi-animal pose tracking. *Nature Methods*, 19, 486-495. doi:10.1038/s41592-022-01426-1.

### Examples

```
# Two flies tracked with SLEAP, in the analysis CSV
path <- system.file("extdata", "sleap.analysis.csv", package = "aniread")
read_sleap(path)

# The same predictions in the analysis .h5, whose frames run to the end of
# the video, and which also carries the skeleton and the SLEAP version
path <- system.file("extdata", "sleap.analysis.h5", package = "aniread")
flies <- read_sleap(path)
flies
anicore::get_structure(flies)
anicore::get_metadata(flies, "source_version")
```

---

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <code>read_structure</code> | <i>Read a skeleton as an anistructure</i> |
|-----------------------------|-------------------------------------------|

---

### Description

#### [Experimental]

Reads the skeleton a pose-estimation project defines — its keypoints and the edges between them — as an `anicore::anistructure()` with points and segments. Attach it to a frame with `anicore::set_structure()`.

- `read_structure_deeplabcut()` reads a DeepLabCut project's `config.yaml`: `bodyparts`, or `multianimalbodyparts` in a multi-animal project, and `skeleton`. A multi-animal project's unique `bodyparts` are left out: DeepLabCut assigns them to a separate "single" individual, not to each animal's body.

- `read_structure_sleap()` reads a SLEAP .slp file (project or predictions) or an analysis .h5 export. Only body edges become segments; SLEAP's symmetry edges pair left and right keypoints and are not segments.
- `read_structure()` detects which from the file, as `read_dataset()` does for frames.

Edges keep the direction the file lists them in, which is what `anicores::as_anisegment()` measures along.

A frame reader attaches whatever its data file itself contains, so whether you need these depends on where the tool keeps its skeleton:

- A SLEAP analysis .h5 carries its skeleton, and `read_sleap()` attaches it as it reads the tracks.
- DeepLabCut keeps its skeleton in the project's config.yaml, not in the data file, so `read_deeplabcut()` attaches none. Read the config with `read_structure()` and attach it with `anicores::set_structure()`.
- SLEAP's analysis CSV carries no skeleton either. Read it from the .slp or .h5 of the same recording.

## Usage

```
read_structure(path, source = "auto")

read_structure_deeplabcut(path)

read_structure_sleap(path, skeleton = NULL)
```

## Arguments

|          |                                                                    |
|----------|--------------------------------------------------------------------|
| path     | Path to the file.                                                  |
| source   | "auto" to detect it from the file, or "deeplabcut" or "sleap".     |
| skeleton | For a .slp holding several skeletons, the name of the one to read. |

## Value

An `anicores::anistructure()`.

## Examples

```
# A DeepLabCut project keeps its skeleton in config.yaml
config <- tempfile(fileext = ".yaml")
writeLines(
  c(
    "bodyparts:",
    "- head",
    "- stinger",
    "skeleton:",
    "- head",
    "- stinger"
  ),
  config
)
```

```

skeleton <- read_structure(config)
skeleton

# Attach it to a frame with the same body parts
path <- system.file("extdata", "deeplabcut.csv", package = "aniread")
read_deeplabcut(path) |>
  anicore::set_structure(skeleton)

# A SLEAP analysis .h5 carries the skeleton of its tracks
path <- system.file("extdata", "sleap.analysis.h5", package = "aniread")
skeleton <- read_structure(path)
skeleton

```

---

|                |                            |
|----------------|----------------------------|
| read_trackball | <i>Read trackball data</i> |
|----------------|----------------------------|

---

## Description

Read trackball data from a variety of setups and configurations.

## Usage

```

read_trackball(
  paths,
  setup = c("of_free", "of_fixed"),
  sampling_rate,
  col_time = "time",
  col_dx = "x",
  col_dy = "y",
  counts_per_rotation = NULL,
  ball_diameter = NULL,
  dots_per_cm = NULL,
  quiet = TRUE
)

```

## Arguments

|               |                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| paths         | Two file paths, one for each sensor (although one is allowed for a fixed setup, of_fixed).                                                                                                                                                        |
| setup         | Which type of experimental setup was used. Expects either of_free or of_fixed.                                                                                                                                                                    |
| sampling_rate | Sampling rate tells the function how long time it should integrate over. A sampling rate of 60(Hz) will mean windows of 1/60 sec are used to integrate over.                                                                                      |
| col_time      | Which column contains the information about time. Can be specified either by the column number (numeric) or the name of the column if it has one (character). Should either be a datetime (POSIXt) or seconds (numeric). <b>With two sensors,</b> |

**this must be a clock the two sensors share.** In a Bonsai capture that is the PC datetime (column 4), not the per-board device counter (column 3), whose origin is sensor-local and therefore cannot be used to cross-reference the two files. A warning is emitted if `col_time` resolves to a non-datetime column with two sensors.

|                                  |                                                                                                                                                                                |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>col_dx</code>              | Column name for x-axis values                                                                                                                                                  |
| <code>col_dy</code>              | Column name for y-axis values                                                                                                                                                  |
| <code>counts_per_rotation</code> | For <code>of_fixed</code> setup: the sensor count for a full 360 degree rotation. Can be obtained using <code>calibrate_trackball()</code> .                                   |
| <code>ball_diameter</code>       | For <code>of_fixed</code> setup: the ball diameter (in same units as desired output). Required if using <code>dots_per_cm</code> instead of <code>counts_per_rotation</code> . |
| <code>dots_per_cm</code>         | For <code>of_fixed</code> setup: sensor dots-per-cm. Use with <code>ball_diameter</code> as an alternative to <code>counts_per_rotation</code> .                               |
| <code>quiet</code>               | If TRUE (default), suppresses informational messages such as the count of malformed rows dropped from each file.                                                               |

## Details

Raw Bonsai optical-flow captures are headerless CSV files, optionally preceded by serial-port junk – a partial row, or a run of noise lines and blanks before the first complete record – with the layout `dx, dy, device_clock_us, pc_datetime, interval_s`. Address their columns by number (`col_dx = 1, col_dy = 2, col_time = 4`).

Such captures are not gap-free: on this rig the COM port emits no row while the ball is still, so multi-second gaps are normal. Readings are integrated into  $1 / \text{sampling\_rate}$  windows, and windows containing no reading are filled with zero motion so the returned time grid is regular regardless of the gaps in the input. Note that this treats a missing sample as *no motion*, which is a property of this logger rather than of optical flow in general; a sensor that drops samples for other reasons would need its gaps treated as missing data instead.

With two sensors the output covers only the intersection of the two recordings - readings from before the second sensor started, or after the first stopped, are discarded rather than zero-filled. `time = 0` is the first shared sample, and the `start_datetime` metadata is the wall-clock instant of that sample.

The sensor has no fixed rate of its own: it reports motion as it happens, and nothing while the ball is still. The frame's `sampling_rate` is therefore the rate of the integration windows, and its `source_sampling_rate` is NaN, anicore's marker for a device with no fixed rate (see `anicore::set_metadata()`), rather than NA, which would mean the rate has not been declared.

## Value

a movement dataframe. `time` is in seconds from the first sample both sensors share; see "Time" in `read_dataset()`. `sampling_rate` is the `sampling_rate` given, the rate of the integration windows, and `source_sampling_rate` is NaN, since the sensor has no fixed rate.

## Examples

```
# A free-floating ball is tracked by two optical flow sensors,
# so both files are supplied together
```

```

paths <- c(
  system.file("extdata", "trackball_sensor_1.csv", package = "aniread"),
  system.file("extdata", "trackball_sensor_2.csv", package = "aniread")
)
read_trackball(paths, setup = "of_free", sampling_rate = 60, col_time = "t")

```

---

read\_trackmate

*Read TrackMate XML into an aniframe*


---

## Description

Parses a TrackMate XML file and returns spot data from filtered tracks as an aniframe. TrackMate stores spot coordinates in image (top-left) coordinates; the reader reflects y so the returned aniframe is in the conventional bottom\_left origin. The frame height is read from Settings/ImageData in the XML by default: height is in pixels and pixelheight in spatialunits, so their product is the height in the unit of the positions. Without a pixelheight, height is used as it is.

## Usage

```
read_trackmate(path, slim = TRUE, video_height = NULL)
```

## Arguments

|              |                                                                                                                                                                 |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path         | Path to the TrackMate XML file.                                                                                                                                 |
| slim         | If TRUE, return only essential columns (default TRUE).                                                                                                          |
| video_height | Optional numeric height of the source frame in the spatial unit reported by the XML. Overrides the height read from Settings/ImageData when both are available. |

## Details

Each track is identified in track by the name TrackMate gave it. TrackMate names tracks Track\_0, Track\_1, ..., and the column already says they are tracks, so a name of that form becomes its number alone: Track\_12 is read as 12. A name you gave a track in TrackMate, anything not of the form Track\_<number>, is kept as written. Each track's name is read on its own, so a file in which you renamed some tracks gives, say, 0, 2 and Cell A. The numeric TRACK\_ID is used instead when any track has no name, or, with a warning, when two tracks would get the same id (such as a track you renamed 3 beside Track\_3). The levels of track are the numbers in numeric order, 2 before 10, followed by any names sorted as text (capitals first, in every locale).

A track that divides, as a cell lineage does, holds more than one spot in a frame from its first division on, which a frame keyed by track and time cannot hold. Such a track is split into its branches, each the stretch of the track between divisions, and each branch becomes a track of its own. The parent column records the lineage, as cell tracking does (the P of the Cell Tracking Challenge's res\_track.txt, Ultrack's parent\_track\_id): it holds the id of the track a track divided from, and is NA for a track that did not divide from another, so for every track of a file without divisions. The branch before the first division keeps the track's id. The other branches are numbered on from the largest number that identifies any track in the file, by TRACK\_ID or by name, so a new id never

names another track you can see in TrackMate. They are numbered track by track, in the order of the tracks' ids, and within a track by the frame they start in, then by the id of the track they divided from, so sisters are numbered together, then by x and y. A daughter starts at its first spot after the division, so nothing is measured across it. A track whose branches also merge, or whose spots share a frame in any other way, is left whole, with a warning about the duplicate track-frame combinations. A split or merge that never puts two spots in one frame, as when a link skips a frame beside another that does not, needs no splitting.

parent is not an identity key: the keys are track and keypoint. It is a factor with the same levels as track, so its values match the ids in track. One row per track gives the lineage, as the Cell Tracking Challenge's L B E P table (label, first and last time, parent). The frame is grouped by its keys, so drop to a plain tibble first:

```
data |>
  dplyr::as_tibble() |>
  dplyr::summarise(
    start = min(time),
    end = max(time),
    parent = dplyr::first(parent),
    .by = track
  )
```

The XML also records how the image was calibrated, and the reader keeps what has a place in the metadata:

- `timeunits` and `spatialunits`, from the Model element, become `unit_time` and `unit_space`. TrackMate takes them from the image's calibration in ImageJ, where they are free text, so the usual spellings are recognised: "sec", "msec", "min", "frame" and so on for time, "pixel", "micron", "um" (with or without the micro sign), "mm" and so on for space. A unit with no equivalent in anicore (days or inches, say) becomes "unknown" or "none", with a warning, rather than stopping the read. A blank spatial unit, which TrackMate writes for an image whose length unit is a space, is read as pixels when the pixel size in Settings/ImageData is 1 or not recorded, as ImageJ calls an image with no length unit "pixel". With any other pixel size the positions are scaled by a unit nobody named, so it becomes "none" with a warning.
- `timeinterval`, from Settings/ImageData, is the time between frames in `timeunits`. Its reciprocal, converted to Hz, becomes `sampling_rate`. It stays NA when the interval is missing or not positive, or when the time unit is frames or not recognised.
- `version`, from the root element, is the TrackMate version that wrote the file and becomes `source_version`.

TrackMate cannot tell an uncalibrated image from one calibrated at one second per frame. An image with no time calibration has a frame interval of 0, which TrackMate replaces with 1, and ImageJ's default time unit is "sec". A file with `timeunits = "sec"` and `timeinterval = 1` may therefore really count frames, in which case time is the frame number and the 1 Hz `sampling_rate` is not the camera's. If you know the real rate, set it with `anicore::set_metadata()`.

## Value

An aniframe with columns including time, track, keypoint, parent, x, y, z and frame.

## References

- Tinevez, J.-Y., Perry, N., Schindelin, J., Hoopes, G. M., Reynolds, G. D., Laplantine, E., Bednarek, S. Y., Shorte, S. L., & Eliceiri, K. W. (2017). TrackMate: An open and extensible platform for single-particle tracking. *Methods*, 115, 80-90. doi:10.1016/j.ymeth.2016.09.016.
- Ershov, D., Phan, M.-S., Pylvänäinen, J. W., et al. (2022). TrackMate 7: integrating state-of-the-art segmentation algorithms into tracking pipelines. *Nature Methods*, 19, 829-832. doi:10.1038/s41592-022-01507-1.

## Examples

```
# A cell that divides into tracks 1 and 2, whose parent is track 0
path <- system.file("extdata", "trackmate.xml", package = "aniread")
data <- read_trackmate(path)
data

# The lineage: one row per track, with the track it divided from
data |>
  dplyr::as_tibble() |>
  dplyr::summarise(
    start = min(time),
    end = max(time),
    parent = dplyr::first(parent),
    .by = track
  )
```

---

read\_trex

---

Read TRex Movement Tracking Data

---

## Description

Reads movement tracking data exported from TRex (Walter & Couzin, 2021), from either of its two exports.

The .npz export is TRex's native one: a zip of .npy arrays, one file per tracked individual, so a whole recording is a vector of paths. It carries the pose keypoints, a per-frame detection probability, the identity, and the recording's frame rate and frame size. The CSV export carries the centroid and midline of a single individual and none of that metadata, so several columns come back NA.

Both exports give time in seconds, so unit\_time is "s". The .npz records the frame rate as frame\_rate, which becomes sampling\_rate. The CSV does not record it, but when it has both a frame and a time column, and every time is the frame number divided by one rate (to within the rounding of time), that rate becomes sampling\_rate: the frames between the first and last row over the seconds between them, or the whole number nearest to it when that fits every row as well. Otherwise, as when the times are camera timestamps that vary from frame to frame, it is left NA.

Both exports give positions in centimetres, so unit\_space is "cm". TRex converts from pixels with its cm\_per\_pixel, the real width of the image (meta\_real\_width) over its width in pixels. When nobody set the real width, TRex assumes 30 cm, so the centimetres are only as real as that setting.

The `.npz` records the frame size as `video_size`, in pixels, and `cm_per_pixel`; their product is the frame's width and height in centimetres, which become the `axis_extents` of `x` and `y` and what `y` is reflected around.

### Usage

```
read_trex(path, format = c("auto", "csv", "npz"), video_height = NULL)
```

### Arguments

|                           |                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>path</code>         | Character string specifying the path to a TRex CSV file. The file should contain columns for: <ul style="list-style-type: none"> <li>• time</li> <li>• x and y coordinates for tracked points (e.g., <code>x_head</code>, <code>y_head</code>)</li> <li>• x and y coordinates for centroid (<code>x_number_wcentroid_cm</code>, <code>y_number_wcentroid_cm</code>)</li> </ul> |
| <code>format</code>       | Which export to read. "auto" (default) reads it from the file: the <code>.npz</code> export is a zip of <code>.npy</code> arrays, the CSV export is not.                                                                                                                                                                                                                       |
| <code>video_height</code> | Optional numeric height of the source video frame in the same spatial units as the tracking output, centimetres. The <code>.npz</code> export records this as <code>video_size</code> and it is used automatically; TRex's CSV export does not, so without it <code>max(y)</code> is used as a fallback when reflecting to <code>bottom_left</code> .                          |

### Details

The function performs several processing steps:

1. Validates the input file format (must be CSV)
2. Reads the data using `vroom` for efficient processing
3. Cleans column names to a consistent format
4. Restructures the data from wide to long format
5. Initializes metadata fields required for movement data

### Value

A data frame containing movement data with the following columns:

- `time`: TRex's time, in seconds, with the first frame of the video at 0; see "Time" in [read\\_dataset\(\)](#)
- `individual`: Factor. The identity TRex assigned, from the `.npz` export; NA from the CSV export, which does not record it
- `keypoint`: Factor identifying tracked points (e.g., "head", "centroid")
- `x`: x-coordinates in centimeters
- `y`: y-coordinates in centimeters
- `confidence`: Numeric. TRex's per-frame detection\_p from the `.npz` export; NA from the CSV export, which does not record it
- `yaw`: TRex's ANGLE, the direction the individual faces, from its posture, when the export includes it. Declared as the frame's orientation, so it is reflected with `y`; the same value on every keypoint of an individual.

## References

- Walter, T., & Couzin, I. D. (2021). TRex, a fast multi-animal tracking system with markerless identification, and 2D estimation of posture and visual fields. *eLife*, 10, e64000. doi:10.7554/eLife.64000.

## See Also

- TRex software: <https://trex.run>

## Examples

```
# TRex's native export: one .npz per tracked individual
path <- system.file("extdata", "trex_id3.npz", package = "aniread")
read_trex(path)

# A whole recording is the vector of paths get_sample_data() returns
read_trex(get_sample_data("trex"))
```

---

|                |                                  |
|----------------|----------------------------------|
| write_aniframe | <i>Write an aniframe to disk</i> |
|----------------|----------------------------------|

---

## Description

This high-level wrapper writes an **aniframe** object to a file in one of the supported formats (parquet, csv, or tsv). We highly recommend using parquet as neither csv or tsv can preserve the metadata.

## Usage

```
write_aniframe(data, filename, ...)
```

## Arguments

|          |                                                                                                                                                                                                                                                                              |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| data     | An <b>aniframe</b> object (see <code>anicore::is_aniframe()</code> ).                                                                                                                                                                                                        |
| filename | Character string specifying the output file name. The file extension determines which writer is used.                                                                                                                                                                        |
| ...      | Additional arguments passed to the format-specific writer ( <code>write_aniframe_csv()</code> or <code>write_aniframe_parquet()</code> ). Typical arguments include <code>delim</code> , <code>col_names</code> , <code>compression</code> , etc., depending on the backend. |

## Details

- Supported extensions** are "parquet", "csv" and "tsv". We highly recommend using parquet as neither csv or tsv can preserve the metadata.
- CSV/TSV files are written with *vroom* for fast I/O: .csv comma-separated and .tsv tab-separated, unless `delim` is passed.
- Parquet files are written with the *arrow* package if installed (install-on-demand if missing).

**Value**

The original data object (invisibly), enabling pipe-friendly usage.

**Examples**

```
## Create a small aniframe for demonstration
df <- anicore::example_anipoint()

## Write the aniframe as CSV
write_aniframe(df, tempfile(fileext = ".csv"))

## Write the same aniframe as Parquet
write_aniframe(df, tempfile(fileext = ".parquet"))
```

---

write\_dataset

---

Write a movement or event dataset to any supported format

---

**Description**

One entry point for every format aniread writes, the counterpart of [read\\_dataset\(\)](#). By default the format is worked out from the file suffix.

**Usage**

```
write_dataset(data, path, format = NULL, by = NULL, ...)
```

**Arguments**

|        |                                                                                                                                                                                                                                                                                                                |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| data   | An aniframe: an <a href="#">anipoint</a> , <a href="#">anievent</a> , or another frame built on them.                                                                                                                                                                                                          |
| path   | Path to write to.                                                                                                                                                                                                                                                                                              |
| format | Which format to write. NULL (the default) infers it from the suffix of path. Otherwise one of the "write" sources in <a href="#">get_supported_sources()</a> .                                                                                                                                                 |
| by     | Key columns to write one file per combination of, or NULL (the default) for a single file. Any of <a href="#">anicore::get_keys()</a> ; the role names "what" and "when" stand for all identity or all temporal keys, so <code>by = c("what", "when")</code> writes one file per track. See "Files per group". |
| ...    | Passed on to the writer for format.                                                                                                                                                                                                                                                                            |

**Details**

`write_dataset()` is a dispatcher, not a new writer: it works out which writer to call and calls it.

When several formats write the same suffix, the first listed in [get\\_supported\\_sources\(\)](#) is inferred: a .csv is written as a plain table by [write\\_aniframe\(\)](#), and inTRACKtive's CSV needs `format = "intracktive"`.

Parquet keeps everything: [read\\_dataset\(\)](#) gives back the same class, grouping and metadata that were written. Other formats keep what the format can hold.

**Value**

data, invisibly.

**Files per group**

With `by`, each file is named from `path` with the key values appended before the suffix: `"mice.csv"` with `by = "individual"` gives `mice_individual-mouse1.csv`, `mice_individual-mouse2.csv`, and so on, and several keys give `mice_individual-mouse1_session-2.csv`.

For other names, put the keys in `path` in braces: `"{session}/mice_{individual}.csv"`. The braces then set `by`, so it can be left out, and directories in `path` are created. Characters other than letters, digits, `.`, `_` and `-` in a value become `-`.

Each file holds an `aniframe` with the same class and metadata as `data`.

**See Also**

[read\\_dataset\(\)](#) to read the file back, [get\\_supported\\_sources\(\)](#) for what can be written, and the individual `write_*`() functions for format-specific arguments.

**Examples**

```
data <- anicore::example_anipoint()
path <- tempfile(fileext = ".parquet")

write_dataset(data, path)
read_dataset(path)

# Name the format where the suffix is shared
write_dataset(data, tempfile(fileext = ".csv"), format = "intracktive")

# One file per individual, or per track
dir <- tempfile()
dir.create(dir)
write_dataset(data, file.path(dir, "mice.parquet"), by = "individual")
write_dataset(data, file.path(dir, "{individual}/{keypoint}.parquet"))
list.files(dir, recursive = TRUE)
```

---

write\_intracktive

*Write aniframe data to inTRACKtive CSV format*

---

**Description**

Converts an `aniframe` to the CSV format required by inTRACKtive for browser-based interactive visualization of tracking data. The function creates unique integer track identifiers from combinations of the `aniframe`'s identity keys, and writes the lineage of dividing tracks.

**Usage**

```
write_intracktive(data, filename, quiet = FALSE)
```

**Arguments**

|          |                                                                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| data     | An aniframe containing tracking data with its index (usually time) and x and y. Optional columns are z for 3D data, parent for lineage and frame for recorded frame numbers (see Details). |
| filename | File path to write the CSV.                                                                                                                                                                |
| quiet    | Suppress messages. TRUE/FALSE. Defaults to FALSE.                                                                                                                                          |

**Details**

inTRACKtive requires tracking data with a unique integer `track_id` for each tracked object. This function numbers the tracks from 1, one for each combination of the aniframe's identity keys (`anicore::get_keys()`), such as individual and keypoint, or track and keypoint for a frame read by `read_trackmate()`.

The output format includes:

- `track_id`: Integer identifier for each unique track
- `t`: The frame number of each row, counted from 0 (see below)
- `x`, `y`: Spatial coordinates
- `z`: Optional third dimension if present
- `parent_track_id`: Only when data has a parent column, as `read_trackmate()` gives a frame with dividing tracks. `parent` names the track a track divided from, and `parent_track_id` is that track's `track_id` (with the same values of the other keys), so inTRACKtive draws the lineage. A track with no parent gets -1, inTRACKtive's value for one, as does a track whose parent is not in data (with a warning). Without a parent column the column is left out, which inTRACKtive reads as no divisions.

inTRACKtive reads `t` as whole frames counted from 0, so the index is written in frames, as `anicore::convert_unit_time()` gives them with "frame":

- A frame whose `unit_time` is "frame" is written as it is. Its times must be whole numbers.
- A frame whose time is in another unit, such as seconds or minutes, is converted to frames with its `sampling_rate`, so that the frame at time 0 is `t = 0`: at 30 Hz, times of 0, 1/30 and 2/30 seconds are written as 0, 1 and 2. A frame with a column named `frame` holding the recorded frame numbers, as `anicore::set_index()` keeps them, is written with those.

The writer stops rather than write frame numbers it would have to invent: when the time is not in frames and no `sampling_rate` is declared, or when the times are not regularly spaced at that rate. Declare the rate with `anicore::set_metadata(data, sampling_rate = )`, or keep the recorded frame numbers in a column named `frame`. `data` itself is not changed.

The resulting CSV can be converted to inTRACKtive's Zarr format using their command-line tools or Python package.

**Value**

Returns the input data unchanged.

**References**

- Huijben, T. A. P. M., Anderson, A. G., Sweet, A., et al. (2025). inTRACKtive: a web-based tool for interactive cell tracking visualization. *Nature Methods*. doi:[10.1038/s41592-025027771](https://doi.org/10.1038/s41592-025027771).

**Examples**

```
# Two flies tracked with SLEAP, one track for each fly and keypoint
flies <- read_sleap(
  system.file("extdata", "sleap.analysis.csv", package = "aniread")
)
path <- tempfile(fileext = ".csv")
write_intractive(flies, path)
head(read.csv(path))
```

# Index

`anicores::anievent()`, [13](#), [14](#)  
`anicores::anipoint()`, [13](#)  
`anicores::anistructure()`, [32](#), [33](#)  
`anicores::as_anisegment()`, [33](#)  
`anicores::convert_unit_time()`, [13](#), [43](#)  
`anicores::get_keys()`, [41](#), [43](#)  
`anicores::set_index()`, [43](#)  
`anicores::set_metadata()`, [9](#), [12](#), [26](#), [29](#), [35](#),  
[37](#)  
`anicores::set_structure()`, [32](#), [33](#)  
`anicores::validate_anievent()`, [13](#)  
`anievent`, [17](#), [41](#)  
`anipoint`, [17](#), [41](#)  
  
`c3dr::c3d_read()`, [15](#)  
`calibrate_trackball`, [2](#)  
  
`detect_source`, [3](#)  
`detect_source()`, [7](#), [17](#), [18](#), [24](#)  
  
`get_sample_data`, [4](#)  
`get_supported_sources`, [7](#)  
`get_supported_sources()`, [4](#), [17](#), [18](#), [41](#), [42](#)  
  
`read_aniframe`, [8](#)  
`read_aniframe()`, [18](#)  
`read_animalta`, [8](#)  
`read_animalta()`, [18](#)  
`read_anipose`, [10](#)  
`read_anipose()`, [18](#)  
`read_bonsai`, [12](#)  
`read_bonsai()`, [18](#)  
`read_boris`, [13](#)  
`read_boris()`, [17](#), [18](#)  
`read_c3d`, [14](#)  
`read_c3d()`, [18](#)  
`read_custom`, [15](#)  
`read_custom()`, [7](#), [18](#)  
`read_dataset`, [16](#)  
  
`read_dataset()`, [3–5](#), [7](#), [8](#), [10–12](#), [14](#), [16](#), [19](#),  
[21](#), [22](#), [24](#), [26](#), [27](#), [29](#), [30](#), [32](#), [33](#), [35](#),  
[39](#), [41](#), [42](#)  
`read_deeplabcut`, [19](#)  
`read_deeplabcut()`, [4](#), [17](#), [18](#), [27](#), [33](#)  
`read_fasttrack`, [20](#)  
`read_fasttrack()`, [18](#)  
`read_fictrac`, [21](#)  
`read_fictrac()`, [18](#)  
`read_freemocap`, [22](#)  
`read_freemocap()`, [18](#)  
`read_idtracker`, [25](#)  
`read_idtracker()`, [17](#), [18](#)  
`read_lightningpose`, [27](#)  
`read_lightningpose()`, [18](#)  
`read_movement`, [28](#)  
`read_movement()`, [18](#)  
`read_octron`, [29](#)  
`read_octron()`, [18](#)  
`read_sleep`, [31](#)  
`read_sleep()`, [18](#), [33](#)  
`read_structure`, [32](#)  
`read_structure()`, [32](#)  
`read_structure_deeplabcut`  
`(read_structure)`, [32](#)  
`read_structure_sleep(read_structure)`,  
[32](#)  
`read_structure_sleep()`, [32](#)  
`read_trackball`, [34](#)  
`read_trackball()`, [3](#), [5](#), [17](#), [18](#)  
`read_trackmate`, [36](#)  
`read_trackmate()`, [18](#), [43](#)  
`read_trex`, [38](#)  
`read_trex()`, [5](#), [18](#)  
  
`tibble`, [7](#)  
  
`write_aniframe`, [40](#)  
`write_aniframe()`, [18](#), [41](#)  
`write_dataset`, [41](#)

`write_dataset()`, [7](#)  
`write_intracktive`, [42](#)