

Package: anicore (via r-universe)

October 14, 2026

Type Package

Title An R package providing core data structures for movement data

Version 0.8.0.9009

Description An R package providing core data structures for movement data.

License MIT + file LICENSE

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

BugReports <https://github.com/animovement/anicore/issues>

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports cli, dplyr, lifecycle, rlang, pillar, anytime

Suggests jsonlite, testthat (>= 3.0.0), tidyr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

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

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-14 15:24:03 UTC

RemoteUrl <https://github.com/animovement/anicore>

RemoteRef HEAD

RemoteSha 56193de0ea88ba7b98292c3cb06d33f942d36f13

Contents

angle_between	3
angle_to_rad	4
anievent	5
anipoint	6
anistructure	8
as_anievent	10
as_anijoint	11
as_anipoint	12
as_anisegment	14
circ_difference	15
circ_mad	15
circ_mean	16
circ_median	17
circ_sd	18
circ_successive_difference	18
convert_inf_to_na	19
convert_nan_to_na	20
convert_unit_angle	20
convert_unit_space	21
convert_unit_time	22
deg_to_rad	24
ensure_is_anievent	24
ensure_is_aniframe	25
ensure_is_anipoint	25
ensure_is_cartesian	26
ensure_is_cartesian_1d	27
ensure_is_cartesian_2d	27
ensure_is_cartesian_3d	28
ensure_is_cylindrical	28
ensure_is_polar	29
ensure_is_spatial	29
ensure_is_spherical	30
example_anipoint	31
example_structure	32
get_angle_direction	32
get_axes	33
get_axis_directions	34
get_coordinate_system	35
get_handedness	35
get_index	36
get_keys	37
get_metadata	37
get_sampling_interval	38
is_anievent	39
is_aniframe	40
is_anijoint	40

is_anipoint	41
is_anisegment	42
is_anistructure	42
is_cartesian	43
is_cartesian_1d	43
is_cartesian_2d	44
is_cartesian_3d	44
is_cylindrical	45
is_polar	45
is_sampling_regular	46
is_spatial	47
is_spherical	48
list_default_metadata	48
metadata_json	49
print.aniframe_metadata	51
print.anistructure	51
rad_to_deg	52
reflect_axis	53
set_axis_directions	54
set_index	55
set_metadata	56
structures	58
to_anievent	59
unwrap_angle	61
validate_anievent	62
validate_anipoint	63
validate_anistructure	64
variables	65
wrap_angle	66
Index	68

angle_between	<i>The angle between two vectors</i>
---------------	--------------------------------------

Description

Row-wise angle between direction vectors, as used for joint angles.

- With no axis, the included angle $\arccos(\mathbf{u} \cdot \mathbf{v} / |\mathbf{u}| |\mathbf{v}|)$, in $[0, \pi]$.
- With an axis, both vectors are projected onto the plane perpendicular to it, and the angle turns from $\mathbf{u}$ to $\mathbf{v}$ about the axis by the right-hand rule, in $[-\pi, \pi]$. For 2D vectors, $c(0, 0, 1)$ gives the signed angle from x toward y .

Vectors need not be unit length. A zero vector, or one parallel to the axis, gives NA.

Usage

```
angle_between(u, v, axis = NULL)
```

Arguments

u, v	Numeric vectors of length 2 or 3, or matrices or data frames with 2 or 3 columns and one row per vector. A single vector is recycled.
axis	NULL, or a vector (or one per row) of length 3.

Value

Numeric vector of angles in radians.

Examples

```
angle_between(c(1, 0), c(0, 1))
angle_between(c(1, 0), c(0, 1), axis = c(0, 0, 1))
angle_between(c(1, 0), c(0, 1), axis = c(0, 0, -1))

# One angle per row
angle_between(rbind(c(1, 0, 0), c(0, 1, 0)), c(0, 0, 1))
```

angle_to_rad	<i>Convert angles between radians and a frame's angular unit</i>
--------------	--

Description

Angular computations work in radians; a frame declares the unit its angles are stored in (`unit_angle`). `angle_to_rad()` reads a frame's angles into radians, and `angle_from_rad()` writes radians back out in the frame's unit, so a function that computes new angles can return them in the unit the frame declares.

These convert values and leave the frame alone. To convert a frame's columns and record the new unit, use `convert_unit_angle()`. Calling that on new columns computed in radians would do nothing in a "deg" frame, since it converts from the unit the frame already declares.

Usage

```
angle_to_rad(x, unit)

angle_from_rad(x, unit)
```

Arguments

x	Numeric vector of angles, or of angular rates.
unit	The angular unit: an aniframe or anievent, whose <code>unit_angle</code> is read, or one of "rad", "deg" or "none". Pass the unit as a string, read once with <code>get_metadata(data, "unit_angle")</code> , to convert inside <code>dplyr::mutate()</code> .

Details

A frame that declares no angular unit ("none") is read as radians: radians are the unit angles are computed in, and such a frame has no other unit to return them in.

Value

Numeric vector, the same length as `x`.

See Also

Other angle utilities: [deg_to_rad\(\)](#), [rad_to_deg\(\)](#), [unwrap_angle\(\)](#), [wrap_angle\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
af <- set_metadata(af, unit_angle = "deg")

# Compute in radians, return in the frame's unit
angle_from_rad(pi / 2, af)

# And read the frame's angles for computing
angle_to_rad(c(0, 90, 180), af)

# With the unit read once, inside mutate()
angle_from_rad(c(0, pi), "deg")
```

anievent

Create an anievent data frame

Description

Creates a specialised data frame for behavioural events in long format: one row per bout (state event) or instant (point event). Like [anipoint\(\)](#), the class inherits the shared `aniframe` substrate — metadata, accessors, dplyr methods — but holds event-bout records rather than per-frame movement data.

Usage

```
anievent(
  ...,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)
```

Arguments

... Name-value pairs to create columns in the data frame.

metadata Optional list of metadata.

<code>variables_what</code>	Character vector of identity columns that together define a unique entity. When NULL (default), auto-detected from a known list (model, individual, subject, track, keypoint).
<code>variables_when</code>	Character vector of temporal columns. When NULL (default), auto-detected from a known grouping list (observation, session, trial) and concatenated with the required temporal endpoints <code>c("start", "stop")</code> .
<code>.rows</code>	Number of rows (passed to tibble).
<code>.name_repair</code>	How to repair column names (passed to tibble).

Details

Mandatory columns: `channel`, `type`, `label`, `start`, `stop`. A "channel" is one mutually-exclusive categorical track of behaviour — at any moment, a subject can only have one label active per channel. `type` is "state" (durative, `stop > start`) or "point" (instant, `start == stop`). Identity columns (e.g. `individual`, `subject`, `track`) are optional and declared via `variables_what`. A `modifiers` list-column may carry per-event modifier values — each cell a character vector (matching the BORIS export format, where one event can have zero or more modifier values selected from the ethogram).

Value

An `anievent` object.

Examples

```
anievent(
  individual = c(1L, 1L, 1L),
  channel = c("behaviour", "behaviour", "call"),
  label = c("REM", "wake", "alarm"),
  start = c(3, 14, 4.5),
  stop = c(9, 19, 4.5)
)
```

anipoint

Create an anipoint data frame

Description

Creates a specialized data frame for movement data with columns defining entity identity, time-points, and spatial position.

Usage

```
anipoint(
  ...,
  metadata = list(),
  variables_what = NULL,
```

```

variables_when = NULL,
variables_where = NULL,
index = NULL,
.rows = NULL,
.name_repair = c("check_unique", "unique", "universal", "minimal")
)

```

Arguments

...	Name-value pairs to create columns in the data frame.
metadata	Optional list of metadata.
variables_what	Character vector of identity columns that together define a unique entity, and which the frame is grouped by. If NULL (the default), detected from the data: whichever of model, individual, subject, track and keypoint are present, in the order <code>list_recognised_variables_what()</code> lists them. Order carries no meaning of its own — see its documentation. An anipoint needs at least one identity variable, so if none of them is found, a keypoint column is added with the value "centroid". Pass <code>character(0)</code> to declare no identity variables at all — a deliberate opt-out, which leaves the frame ungrouped. Every column named here must exist in data.
variables_when	Character vector of temporal columns that together define a unique timepoint. If NULL (the default), detected from the data: whichever of observation, session, trial and time are present, minus the index. These are the temporal <i>context</i> — which session, which trial — and, together with <code>variables_what</code> , they are what the frame is grouped by. The index itself is declared separately and is never one of them.
variables_where	The spatial columns that together define position. Either a plain character vector of column names, in which case the name is taken to be the axis role, or a vector named by axis role — <code>c(x = "u", y = "v")</code> — which lets the columns be called anything. The roles themselves are a closed set (x, y, z, rho, phi, theta), so that transformations between coordinate systems stay well defined; an unrecognised role is rejected by name. If NULL (the default), detected from the data.
index	Length-one character vector naming the column the frame is indexed by — the position of each row within its temporal context. It is never a grouping variable. If NULL (the default), the frame's existing declaration is kept, or "time" for a frame that has none. The column must exist and be numeric; it may be called anything.
.rows	Number of rows (passed to tibble).
.name_repair	How to repair column names (passed to tibble).

Value

An anipoint object (tibble with anipoint class).

Examples

```
anipoint(
  individual = rep(1:2, each = 25),
  time = rep(1:10, 5),
  x = rnorm(50),
  y = rnorm(50)
)

# Custom variables
anipoint(
  track = rep(1:3, each = 10),
  trial = 1,
  time = rep(1:10, 3),
  x = rnorm(30),
  y = rnorm(30),
  variables_what = "track",
  variables_when = "trial"
)

# Indexed by a column that isn't called `time`
anipoint(
  individual = 1L,
  frame = 1:10,
  x = rnorm(10),
  y = rnorm(10),
  index = "frame"
)
```

anisttructure

Create a structure: points, the segments between them, and joints

Description

[Experimental]

An anisttructure describes how the levels of one identity or temporal variable relate: the keypoints of a skeleton, or the players of a team. It has three parts, each optional beyond the first:

- **points** — the members, levels of the variable it is attached to.
- **segments** — directed links between two points, with an optional expected length. Direction matters for joint angles: run limbs proximal to distal, and paired segments (left to right) the same way.
- **joints** — one measured angle each, between an ordered pair of segments (a, b), with optional limits min, max and rest in radians. The angle is 0 when the two segments point the same way. In 2D it is signed, turning from a to b. In 3D it is the included angle, or, when axis is given ("x", "y", "z" or a segment name), the signed angle about it. Several angles on the same pair of segments — flexion and abduction, say — are several joints with different axes.

Lengths and limits are recorded, not enforced: checks compare the data against them.

A structure is a template: it records no data and no variable until it is attached to a frame with `set_structure()`, so one structure can be reused across frames.

Usage

```
anisttructure(
  points = NULL,
  segments = NULL,
  joints = NULL,
  root = NA_character_,
  source = NA_character_,
  citation = NA_character_,
  license = NA_character_
)
```

Arguments

<code>points</code>	Character vector of point names. Defaults to the segment endpoints.
<code>segments</code>	A data frame with <code>from</code> and <code>to</code> , and optionally <code>segment</code> (defaults to "from-to") and <code>length</code> ; or a list of <code>c(from, to)</code> pairs.
<code>joints</code>	A data frame with <code>a</code> and <code>b</code> (segment names), and optionally <code>joint</code> (defaults to "a-b"), <code>axis</code> , <code>min</code> , <code>max</code> and <code>rest</code> .
<code>root</code>	The point the structure hangs from when positions are rebuilt from segments. NA if unset.
<code>source</code> , <code>citation</code> , <code>license</code>	Provenance of the structure.

Value

An anisttructure.

See Also

[set_structure\(\)](#), [example_structure\(\)](#)

Examples

```
anisttructure(
  segments = data.frame(
    segment = c("thigh", "shin"),
    from = c("hip", "knee"),
    to = c("knee", "ankle")
  ),
  joints = data.frame(joint = "knee", a = "thigh", b = "shin", min = 0, max = 2.5),
  root = "hip"
)

# A team: points only, or points with links between them
anisttructure(points = c("gk", "lb", "cb", "rb"))
anisttructure(segments = list(c("lb", "cb"), c("cb", "rb")))
```

as_anievent	<i>Cast a data frame to an anievent</i>
-------------	---

Description

Strict cast — the input must already be in canonical anievent shape: one row per bout or instant, with the columns channel, type, label, start, and stop. Identity columns travel via `variables_what`. An optional `modifiers` list-column may carry per-event modifier values (each cell a character vector, matching the BORIS export format).

Usage

```
as_anievent(
  data,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL
)

## S3 method for class 'anievent'
as_anievent(
  data,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL
)

## S3 method for class 'aniframe'
as_anievent(
  data,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL
)

## S3 method for class 'data.frame'
as_anievent(
  data,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL
)
```

Arguments

<code>data</code>	A data frame with the required columns.
<code>metadata</code>	Optional list of metadata.

- variables_what** Character vector of identity columns. When NULL (default), auto-detected from a known list (model, individual, subject, track, keypoint) — only those present in data are used. Pass an explicit value to use any other column name(s) as identity. An anievent with no identity column is permitted (e.g. a single-subject experiment).
- variables_when** Character vector of temporal columns. When NULL (default), auto-detected from a known grouping list (observation, session, trial) and concatenated with the required temporal endpoints c("start", "stop"). Pass explicitly to use other names for the grouping context.

Details

To *encode* per-frame data (factor / logical / character columns) into the bout shape, use [to_anievent\(\)](#) instead.

Value

An anievent object.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
try(as_anievent(af))
```

as_anijoint

Convert segments to joints: one angle per joint

Description

[Experimental]

Computes the angle of each joint of a structure from the directions of its two segments a and b, with [angle_between\(\)](#):

- In 2D, the signed angle turning from a to b, in $[-\pi, \pi]$, positive from x toward y. [get_angle_direction\(\)](#) says how that looks on screen.
- In 3D with no joint axis, the included angle, in $[0, \pi]$.
- In 3D with an axis ("x", "y", "z" or a segment name), the signed angle about it after projecting both segments onto the plane perpendicular to it.

The angle is 0 when the two segments point the same way. The segment key is replaced by a joint key, the angle is in the frame's unit_angle, and confidence is the lower of the two segments'.

A joint frame cannot be converted back: it keeps no lengths and, in 3D, no rotation about the segments themselves.

Usage

```
as_anijoint(data, structure = NULL)
```

Arguments

data	An <code>as_anisegment()</code> frame, or an anipoint, which is converted to segments first.
structure	For an anipoint, the structure to use; see <code>as_anisegment()</code> .

Value

An anijoint.

See Also

`angle_between()`, `anistructure()`

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1) |>
  set_structure(example_structure())
as_anijoint(af)
```

as_anipoint

Convert a data frame to anipoint

Description

Convert a data frame to anipoint

Usage

```
as_anipoint(
  data,
  metadata = list(),
  variables_what = NULL,
  variables_when = NULL,
  variables_where = NULL,
  index = NULL,
  root = NULL
)
```

Arguments

data	A data frame with movement data.
metadata	A list of metadata to attach to the anipoint.

variables_what	Character vector of identity columns that together define a unique entity, and which the frame is grouped by. If NULL (the default), detected from the data: whichever of model, individual, subject, track and keypoint are present, in the order <code>list_recognised_variables_what()</code> lists them. Order carries no meaning of its own — see its documentation. An anipoint needs at least one identity variable, so if none of them is found, a keypoint column is added with the value "centroid". Pass <code>character(0)</code> to declare no identity variables at all — a deliberate opt-out, which leaves the frame ungrouped. Every column named here must exist in data.
variables_when	Character vector of temporal columns that together define a unique timepoint. If NULL (the default), detected from the data: whichever of observation, session, trial and time are present, minus the index. These are the temporal <i>context</i> — which session, which trial — and, together with <code>variables_what</code> , they are what the frame is grouped by. The index itself is declared separately and is never one of them.
variables_where	The spatial columns that together define position. Either a plain character vector of column names, in which case the name is taken to be the axis role, or a vector named by axis role — <code>c(x = "u", y = "v")</code> — which lets the columns be called anything. The roles themselves are a closed set (x, y, z, rho, phi, theta), so that transformations between coordinate systems stay well defined; an unrecognised role is rejected by name. If NULL (the default), detected from the data.
index	Length-one character vector naming the column the frame is indexed by — the position of each row within its temporal context. It is never a grouping variable. If NULL (the default), the frame's existing declaration is kept, or "time" for a frame that has none. The column must exist and be numeric; it may be called anything.
root	[Experimental] For an <code>as_anisegment()</code> frame: an anipoint holding the root point's trajectory, such as the frame the segments came from. The other points are rebuilt by walking the structure's segments outward from it, so edits to the segments (such as constant lengths) carry into the positions.

Value

An anipoint object

Examples

```
df <- data.frame(
  time = 1:3, individual = 'a', keypoint = 'centroid',
  x = c(0, 1, 2), y = c(0, 1, 0)
)
as_anipoint(df)
```

as_anisegment

*Convert an anipoint to segments: a length and a direction per segment***Description****[Experimental]**

Re-expresses the positions of a structure's points as its segments. Each row is one segment at one time: its length and the unit vector ux , uy (and uz in 3D) pointing from the segment's from point to its to point, in the frame's axes and units.

The structure's variable (usually keypoint) is replaced by a segment key. Structures over that variable other than structure are dropped; those over the remaining keys are kept. A confidence column becomes the lower of the two endpoints' values; other undeclared columns are dropped.

[as_anipoint\(\)](#) rebuilds positions from the segments, anchored at the root point's trajectory. That is useful after editing the segments — for example holding each segment's length constant — since every point but the root then moves to agree with them.

Usage

```
as_anisegment(data, structure = NULL)
```

Arguments

data	A Cartesian 2D or 3D anipoint.
structure	Name of the structure to use. May be omitted when only one structure has segments.

Value

An anisegment.

See Also

[anisttructure\(\)](#), [set_structure\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1) |>
  set_structure(example_structure())
seg <- as_anisegment(af)
seg

# Hold each segment's length constant, then rebuild the positions
rigid <- seg |>
  dplyr::mutate(length = stats::median(length))
rebuilt <- as_anipoint(rigid, root = af)
```

circ_difference	<i>Shortest signed distance between two angles</i>
-----------------	--

Description

The difference $\text{to_angle} - \text{from_angle}$, wrapped to $(-\pi, \pi]$ so that it is the shorter way round the circle rather than the arithmetic difference. This is the primitive the circular summaries are built on.

Usage

```
circ_difference(from_angle, to_angle)
```

Arguments

from_angle	A numeric vector of angles, in radians.
to_angle	A numeric vector of angles, in radians.

Value

Numeric vector of signed angular differences in $(-\pi, \pi]$, positive anticlockwise.

See Also

Other circular statistics: [circ_mad\(\)](#), [circ_mean\(\)](#), [circ_median\(\)](#), [circ_sd\(\)](#), [circ_successive_difference\(\)](#)

Examples

```
# a tenth of a turn, not nine tenths
circ_difference(0.1, 6.1)

circ_difference(c(0, pi / 2), c(pi / 2, 0))
```

circ_mad	<i>Circular median absolute deviation</i>
----------	---

Description

The median of the angular distances from the circular median.

Usage

```
circ_mad(x, na_rm = TRUE)
```

Arguments

- x A numeric vector of angles, in radians, in any range.
- na_rm A logical value (default TRUE) determining whether missing values are removed before computing. When FALSE, any NA gives NA.

Value

A single number in radians, in $[0, \pi]$, or NA_real_ when there is nothing to summarise.

See Also

Other circular statistics: [circ_difference\(\)](#), [circ_mean\(\)](#), [circ_median\(\)](#), [circ_sd\(\)](#), [circ_successive_difference\(\)](#)

Examples

```
circ_mad(c(0.1, 0.2, 6.2))
```

circ_mean	<i>Circular mean</i>
-----------	----------------------

Description

The mean direction: the angle of the vector sum of the unit vectors pointing along each observation.

Usage

```
circ_mean(x, na_rm = TRUE)
```

Arguments

- x A numeric vector of angles, in radians, in any range.
- na_rm A logical value (default TRUE) determining whether missing values are removed before computing. When FALSE, any NA gives NA.

Details

The mean is signed, in $(-\pi, \pi]$, the range the suite uses for every direction (see [wrap_angle\(\)](#)), so it can be compared directly with the angles it summarises. Wrap it with `wrap_angle(x, "2pi")` for $[0, 2\pi)$.

Value

A single angle in $(-\pi, \pi]$, or NA_real_ when there is nothing to summarise.

See Also

Other circular statistics: [circ_difference\(\)](#), [circ_mad\(\)](#), [circ_median\(\)](#), [circ_sd\(\)](#), [circ_successive_difference\(\)](#)

Examples

```
# 10 degrees; an arithmetic mean would say 190
rad_to_deg(circ_mean(deg_to_rad(c(350, 30))))

# -10 degrees, in the range of the angles themselves
rad_to_deg(circ_mean(deg_to_rad(c(-30, 10))))
```

circ_median

*Circular median***Description**

Fisher's circular median: the direction minimising the summed angular distance to every observation. Where two directions tie, their mean direction is returned.

Usage

```
circ_median(x, na_rm = TRUE)
```

Arguments

x	A numeric vector of angles, in radians, in any range.
na_rm	A logical value (default TRUE) determining whether missing values are removed before computing. When FALSE, any NA gives NA.

Details

The median is signed, in $(-\pi, \pi]$, the range the suite uses for every direction (see [wrap_angle\(\)](#)), so it can be compared directly with the angles it summarises. Wrap it with `wrap_angle(x, "2pi")` for $[0, 2\pi)$.

Value

A single angle in $(-\pi, \pi]$, or `NA_real_` when there is nothing to summarise.

See Also

Other circular statistics: [circ_difference\(\)](#), [circ_mad\(\)](#), [circ_mean\(\)](#), [circ_sd\(\)](#), [circ_successive_difference\(\)](#)

Examples

```
# a little below zero, not a little below 2 * pi
circ_median(c(-0.1, -0.2, 0.05))

# unaffected by where the circle is cut
circ_median(c(0.1, 0.2, 6.2))
circ_median(c(0.1, 0.2, 6.2) + pi)
```

circ_sd	<i>Circular standard deviation</i>
---------	------------------------------------

Description

Computed from the mean resultant length as $\sqrt{-2 * \log(R)}$, so it grows without bound as the angles spread out rather than saturating at π .

Usage

```
circ_sd(x, na_rm = TRUE)
```

Arguments

x	A numeric vector of angles, in radians, in any range.
na_rm	A logical value (default TRUE) determining whether missing values are removed before computing. When FALSE, any NA gives NA.

Value

A single non-negative number in radians, 0 when every angle is the same, or NA_real_ when there is nothing to summarise.

See Also

Other circular statistics: [circ_difference\(\)](#), [circ_mad\(\)](#), [circ_mean\(\)](#), [circ_median\(\)](#), [circ_successive_difference\(\)](#)

Examples

```
circ_sd(c(0.1, 0.2, 0.15))

# identical angles have no spread
circ_sd(rep(1.3, 5))
```

circ_successive_difference	<i>Differences between successive angles in a series</i>
----------------------------	--

Description

Applies [circ_difference\(\)](#) along a vector, comparing each angle with the one lag positions before it — the turn from one heading to the next, rather than the difference between two angles you name. Unlike [base::diff\(\)](#) the result is the same length as x, padded with NA at the start, so it can be used inside [dplyr::mutate\(\)](#).

Usage

```
circ_successive_difference(x, lag = 1L)
```

Arguments

x A numeric vector of angles, in radians.

lag A positive integer (default 1L) giving the lag to difference at.

Value

A numeric vector the same length as **x**, in radians. The first **lag** entries are NA; the rest are angular differences in $(-\pi, \pi]$.

See Also

Other circular statistics: [circ_difference\(\)](#), [circ_mad\(\)](#), [circ_mean\(\)](#), [circ_median\(\)](#), [circ_sd\(\)](#)

Examples

```
circ_successive_difference(c(0, pi / 2, pi, 3 * pi / 2))

# crossing zero is a small step, not a large one
circ_successive_difference(c(6.2, 0.1))
```

convert_inf_to_na	<i>Convert Inf to NA in numeric columns</i>
-------------------	---

Description

Replaces all Inf and -Inf values with NA in numeric columns of a data frame. The sibling of [convert_nan_to_na\(\)](#), for sources that mark a missing observation with an infinity rather than a NaN — TRex is one, and its own documentation masks `np.inf` out before plotting.

Usage

```
convert_inf_to_na(data)
```

Arguments

data A data frame.

Details

Worth doing at read time rather than later: an Inf propagates through arithmetic silently, so a single untracked frame turns a mean, a speed or a bounding box into Inf rather than into a missing value.

Value

A data frame with Inf and -Inf replaced by NA in numeric columns.

Examples

```
df <- data.frame(x = c(1, Inf, -Inf, 3))
convert_inf_to_na(df)
```

convert_nan_to_na	<i>Convert NaN to NA in numeric columns</i>
-------------------	---

Description

Replaces all NaN values with NA in numeric columns of a data frame.

Usage

```
convert_nan_to_na(data)
```

Arguments

data A data frame.

Value

A data frame with NaN values replaced by NA in numeric columns.

Examples

```
df <- data.frame(x = c(1, NaN, 3))
convert_nan_to_na(df)
```

convert_unit_angle	<i>Convert the angular unit of an anipoint</i>
--------------------	--

Description

Converts angular columns between radians and degrees and records the new unit_angle. The spatial angular columns phi and theta and a declared yaw are always converted; other angular columns are named in cols.

To declare a unit without changing values, use set_metadata(data, unit_angle = "deg").

Usage

```
convert_unit_angle(data, to_unit, cols = NULL)
```

Arguments

data	An anipoint, or an anijoint, whose angle is converted.
to_unit	"rad" or "deg".
cols	Further numeric angular columns to convert.

Value

data, converted, with unit_angle updated.

Examples

```
df <- data.frame(time = 1:3, rho = 1:3, phi = c(0, pi / 2, pi))
convert_unit_angle(as_anipoint(df), "deg")
```

convert_unit_space	<i>Convert the spatial unit of an anipoint</i>
--------------------	--

Description

Rescales the columns carrying a length and records the new unit_space. Those are the length axes of the coordinate system — x, y, z on a Cartesian frame, rho (and z) on a polar, cylindrical or spherical one. Angular axes are `convert_unit_angle()`'s. Declared axis_extents are rescaled with them.

To declare a unit without changing values, use `set_metadata(data, unit_space = "mm")`.

Usage

```
convert_unit_space(data, to_unit, calibration_factor = NULL)
```

Arguments

data	An anipoint, or an anisegment, whose length is rescaled.
to_unit	Target unit, one of the levels of unit_space in <code>list_default_metadata()</code> .
calibration_factor	Multiplier from the current unit to to_unit. Derived between metric units; required when converting from "px".

Value

data, rescaled, with unit_space updated.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)

# 1 px = 0.5 mm
af_mm <- convert_unit_space(af, "mm", calibration_factor = 0.5)
convert_unit_space(af_mm, "cm")
```

convert_unit_time	<i>Convert the time unit of an anipoint or anievent</i>
-------------------	---

Description

Rescales the temporal columns — the index of an anipoint, start and stop of an anievent — and records the new unit_time. An anipoint's sampling_interval is measured again in the new unit; sampling_rate is in Hz and is left as it is. Between SI units the factor is derived; to and from "frame" it is derived from the declared sampling_rate.

Frames count from 0: the first frame is at time 0, so frames are seconds multiplied by the rate, and seconds are frames divided by it. Converting from frames to a time unit keeps nothing extra, as frames are evenly spaced and multiplying by the rate gives them back.

Converting back to frames is also how a wrong rate is put right: convert to "frame" with the rate the frame declares, declare the right one, and convert forward again (see the examples). When it is the camera's rate that was wrong, declare it as source_sampling_rate too, which records the rate the device recorded at (see [set_metadata\(\)](#)).

To declare a unit without changing values, use `set_metadata(data, unit_time = "s")`.

Usage

```
convert_unit_time(data, to_unit, calibration_factor = NULL)

## S3 method for class 'anipoint'
convert_unit_time(data, to_unit, calibration_factor = NULL)

## S3 method for class 'anisegment'
convert_unit_time(data, to_unit, calibration_factor = NULL)

## S3 method for class 'anijoint'
convert_unit_time(data, to_unit, calibration_factor = NULL)

## S3 method for class 'anievent'
convert_unit_time(data, to_unit, calibration_factor = NULL)
```

Arguments

data	An anipoint or anievent.
------	--------------------------

to_unit	Target unit, one of the levels of unit_time in <code>list_default_metadata()</code> other than "unknown".
calibration_factor	Multiplier from the current unit to to_unit. Required when converting to or from "frame" without a sampling_rate, or from "unknown". Refused when the frame numbers are taken from a frame column.

Value

data, rescaled, with unit_time and, for an anipoint, sampling_interval updated.

Converting to frames

Frame numbers that a file records are data; frame numbers computed from a rate are only nominal. So converting to "frame" uses recorded frame numbers when the frame has them, and computes them only from regular sampling.

- **Recorded frame numbers.** If an anipoint has a column named frame that is not its index, it holds the frame numbers recorded with the data, and converting to "frame" makes that column the index again rather than computing frames from the rate. The column it replaces stays in the frame as an ordinary column, so the times are not lost. `set_index()` puts the frame numbers there when it moves the index from frames to another column, such as recorded timestamps.
- **Computed frame numbers.** Otherwise each time is multiplied by the rate and rounded to the nearest whole frame. Rather than invent frame numbers, the conversion refuses when the sampling is irregular, with a gap between consecutive times of the same keys more than 1% (the margin `validate_anipoint()` allows by default) from a whole number of frames, or when rounding would put two different times of the same keys on the same frame. A timestamp log, or a sampling_rate that does not match the data, is refused this way.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
af <- set_metadata(af, sampling_rate = 30)
in_seconds <- convert_unit_time(af, "s")
in_seconds

# The camera was really 25 fps: back to frames, then forward at 25 fps
in_seconds |>
  convert_unit_time("frame") |>
  set_metadata(sampling_rate = 25, source_sampling_rate = 25) |>
  convert_unit_time("s")

# Times off the frame grid by a little jitter round to whole frames
jittered <- dplyr::mutate(in_seconds, time = time + c(1e-4, -1e-4, 0))
convert_unit_time(jittered, "frame")$time
```

deg_to_rad	<i>Convert degrees to radians</i>
------------	-----------------------------------

Description

Convert degrees to radians

Usage

```
deg_to_rad(x)
```

Arguments

x	Numeric vector of angles (degrees).
---	-------------------------------------

Value

Numeric vector of angles expressed in radians.

See Also

Other angle utilities: [angle_to_rad\(\)](#), [rad_to_deg\(\)](#), [unwrap_angle\(\)](#), [wrap_angle\(\)](#)

Examples

```
deg_to_rad(180)
deg_to_rad(c(0, 90, 180))
```

ensure_is_anievent	<i>Ensure object is an anievent</i>
--------------------	-------------------------------------

Description

Ensure object is an anievent

Usage

```
ensure_is_anievent(x)
```

Arguments

x	An object to test.
---	--------------------

Value

Errors if x is not an anievent; otherwise returns invisibly.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
try(ensure_is_anievent(af))
```

ensure_is_aniframe	<i>Ensure object is an aniframe</i>
--------------------	-------------------------------------

Description

Guard form of `is_aniframe()`, passed by any animovement frame class. Functions that need coordinates should guard with `ensure_is_anipoint()` instead.

Usage

```
ensure_is_aniframe(x)
```

Arguments

x	An object to test
---	-------------------

Value

Error if not an aniframe

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently, and errors otherwise
ensure_is_aniframe(af)

try(ensure_is_aniframe(data.frame(x = 1)))
```

ensure_is_anipoint	<i>Ensure object is an anipoint</i>
--------------------	-------------------------------------

Description

Ensure object is an anipoint

Usage

```
ensure_is_anipoint(x)
```

Arguments

x	An object to test
---	-------------------

Value

Error if not an anipoint

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently, and errors otherwise
ensure_is_anipoint(af)

try(ensure_is_anipoint(data.frame(x = 1)))
```

ensure_is_cartesian	<i>Ensure an anipoint uses a Cartesian coordinate system</i>
---------------------	--

Description

Stops with an informative error if data is not in a Cartesian coordinate system.

Usage

```
ensure_is_cartesian(data)
```

Arguments

data	An anipoint.
------	--------------

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently when the coordinate system matches
ensure_is_cartesian(af)
```

`ensure_is_cartesian_1d`*Ensure an anipoint uses a 1-D Cartesian coordinate system*

Description

Stops with an informative error if data is not in a 1-D Cartesian coordinate system.

Usage

```
ensure_is_cartesian_1d(data)
```

Arguments

`data` An anipoint.

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
try(ensure_is_cartesian_1d(af))
```

`ensure_is_cartesian_2d`*Ensure an anipoint uses a 2-D Cartesian coordinate system*

Description

Stops with an informative error if data is not in a 2-D Cartesian coordinate system.

Usage

```
ensure_is_cartesian_2d(data)
```

Arguments

`data` An anipoint.

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
ensure_is_cartesian_2d(af)
```

```
ensure_is_cartesian_3d
```

Ensure an anipoint uses a 3-D Cartesian coordinate system

Description

Stops with an informative error if data is not in a 3-D Cartesian coordinate system.

Usage

```
ensure_is_cartesian_3d(data)
```

Arguments

data An anipoint.

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
try(ensure_is_cartesian_3d(af))
```

```
ensure_is_cylindrical
```

Ensure an anipoint uses a cylindrical coordinate system

Description

Stops with an informative error if data is not in a cylindrical coordinate system.

Usage

```
ensure_is_cylindrical(data)
```

Arguments

data An anipoint.

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently when the coordinate system matches
try(ensure_is_cylindrical(af))
```

ensure_is_polar	<i>Ensure an anipoint uses a polar coordinate system</i>
-----------------	--

Description

Stops with an informative error if data is not in a polar coordinate system.

Usage

```
ensure_is_polar(data)
```

Arguments

data An anipoint.

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently when the coordinate system matches
try(ensure_is_polar(af))
```

ensure_is_spatial	<i>Ensure the spatial columns match the metadata</i>
-------------------	--

Description

Guard form of `is_spatial()`, for functions that reach coordinates by iterating `variables_where`. Aborts naming the offending columns, so the error points at the metadata mismatch rather than surfacing later and further away.

Usage

```
ensure_is_spatial(data)
```

Arguments

data An anipoint object.

Value

The input data, invisibly.

See Also

`is_spatial()`, `validate_anipoint()`.

Examples

```
af <- anipoint(time = 1:5, x = 1:5, y = 1:5)
ensure_is_spatial(af)
```

ensure_is_spherical	<i>Ensure an anipoint uses a spherical coordinate system</i>
---------------------	--

Description

Stops with an informative error if data is not in a spherical coordinate system.

Usage

```
ensure_is_spherical(data)
```

Arguments

data	An anipoint.
------	--------------

Value

TRUE, invisibly, if the coordinate system matches; otherwise an error.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
# Passes silently when the coordinate system matches
try(ensure_is_spherical(af))
```

example_anipoint	<i>Create example anipoint data</i>
------------------	-------------------------------------

Description

Generates a synthetic anipoint object with random coordinates for testing and demonstration purposes. The function creates a complete design with all combinations of time points, individuals, keypoints, trials, and sessions.

Usage

```
example_anipoint(
  n_obs = 50,
  n_individuals = 3,
  n_keypoints = 11,
  n_trials = 1,
  n_sessions = 1,
  n_dims = 2
)
```

Arguments

n_obs	Integer. Number of time observations per combination. Default is 50.
n_individuals	Integer. Number of individuals to simulate. Default is 3.
n_keypoints	Integer. Number of keypoints per individual (max 11). Default is 11. When set to 1, only "centroid" is used. Otherwise, anatomical keypoints are used (head, neck, shoulders, etc.).
n_trials	Integer. Number of trials per session. Default is 1.
n_sessions	Integer. Number of sessions. Default is 1.
n_dims	Integer. Number of spatial dimensions (1, 2, or 3). Default is 2. If 1, only x coordinates are generated. If 2, x and y coordinates are generated. If 3, x, y, and z coordinates are generated.

Value

An anipoint object containing randomly generated tracking data with columns for individual, keypoint, time, trial, session, and spatial coordinates (x, y, and/or z depending on n_dims). The coordinates are drawn from a standard normal distribution.

Examples

```
# Create a basic example with default parameters (2D)
example_anipoint()

# Create a 1D example
example_anipoint(n_dims = 1)
```

```
# Create a 3D example
example_anipoint(n_dims = 3)

# Create a smaller example with 2 individuals and 5 keypoints
example_anipoint(n_individuals = 2, n_keypoints = 5)

# Create example with multiple trials and sessions
example_anipoint(n_obs = 100, n_trials = 3, n_sessions = 2)

# Create minimal example with just centroid in 3D
example_anipoint(n_keypoints = 1, n_dims = 3)
```

example_structure	<i>An example structure for the keypoints of example_anipoint()</i>
-------------------	---

Description

A spine from abdomen to neck, a head, shoulders, and two legs with knee joints. Limits are illustrative, not anatomical.

Usage

```
example_structure()
```

Value

An anisttructure.

Examples

```
example_structure()
```

get_angle_direction	<i>Which way angles run</i>
---------------------	-----------------------------

Description

The sense of rotation from the x axis to the y axis, as seen from where the recording was made. `atan2(y, x)` counts counter-clockwise, so a frame stored the other way up reports the mirror of the angle a counter_clockwise frame would give for the same physical heading.

Derived from [get_axis_directions\(\)](#) rather than recorded, so it cannot go on claiming a sense the axes no longer have.

Usage

```
get_angle_direction(data)
```

Arguments

data An aniframe or anievent object.

Value

"clockwise", "counter_clockwise", or "unknown" when the two axes are not both declared or do not span the view.

See Also

[get_axis_directions\(\)](#), [get_handedness\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)

# An image-plane frame counts angles clockwise
af <- set_axis_directions(af, c(x = "right", y = "down"))
get_angle_direction(af)
```

get_axes

The axis roles of an anipoint, and the columns carrying them

Description

The where\$position slot. Index by role to write a transformation that does not care what the columns are called: data[[get_axes(data)[["x"]]]].

Usage

```
get_axes(data)
```

Arguments

data An anipoint object.

Value

Named character vector: names are axis roles (x, y, z, rho, phi, theta), values are the columns carrying them. Empty for a frame whose coordinate system is "unknown".

See Also

[set_variables\(\)](#) to change it.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
get_axes(af)

df <- data.frame(time = 1:3, individual = "a", u = c(1, 2, 3), v = c(0, 1, 0))
renamed <- as_anipoint(df, variables_where = c(x = "u", y = "v"))
get_axes(renamed)
get_metadata(renamed, "coordinate_system")
```

get_axis_directions	<i>Get the direction each axis points</i>
---------------------	---

Description

Get the direction each axis points

Usage

```
get_axis_directions(data)
```

Arguments

data An aniframe or anievent object.

Value

Named character vector, axis role to direction. Empty when the frame declares none.

See Also

[set_axis_directions\(\)](#), [reflect_axis\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
af <- set_axis_directions(af, c(x = "right", y = "up"))
get_axis_directions(af)
```

get_coordinate_system *The coordinate system an anipoint is in*

Description

Derived from the axis roles rather than declared: `set_variables()` says which column carries which role, and the system follows from the set of roles present. It is therefore not writable — see `set_variables()` to say what the columns mean, or `anispac`'s `map_to_*`() functions to convert the coordinates themselves.

Usage

```
get_coordinate_system(data)
```

Arguments

data An anipoint or anievent object.

Value

Length-one character vector: one of "cartesian_1d", "cartesian_2d", "cartesian_3d", "polar", "cylindrical", "spherical" or "unknown".

See Also

`get_axes()`, `is_cartesian()`, `is_polar()`

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
get_coordinate_system(af)
```

get_handedness *Whether the frame is right- or left-handed*

Description

Three declared axis directions determine it, and are read in preference to anything recorded. A frame that states the convention without spelling the axes out — most 3D recordings — declares it with `set_metadata(data, handedness = "right")`.

Usage

```
get_handedness(data)
```

Arguments

data An aniframe or anievent object.

Value

"right", "left", or "unknown" when neither the axes nor the frame itself says.

See Also

[get_axis_directions\(\)](#), [get_angle_direction\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)

# Two axes are not enough
af <- set_axis_directions(af, c(x = "right", y = "up"))
get_handedness(af)
```

get_index

The column an anipoint is indexed by

Description

Exactly one column, of any name, holding the position of each row within its temporal context: the when\$index slot. It is never a grouping column. An [anievent\(\)](#) has none, since a bout spans the start/stop interval.

Usage

```
get_index(data)
```

Arguments

data An anipoint object.

Value

Length-one character vector naming the index column.

See Also

[set_index\(\)](#) to change it, [get_variables\(\)](#) for the other temporal columns.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
get_index(af)
```

`get_keys`*The grouping columns of a frame*

Description

The frame is grouped by its identity keys and temporal context keys: `c(what$keys, when$keys)`. The index and an anievent's interval are never grouping columns.

Usage

```
get_keys(data)
```

Arguments

`data` An aniframe.

Value

Character vector of column names.

See Also

[get_variables\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
get_keys(af)
```

`get_metadata`*Get metadata*

Description

The metadata is stored as a category tree (see [list_default_metadata\(\)](#)), but lookup stays flat: a field is found by its own name wherever it lives, and a category name returns the whole category. The variables category is the exception — its slots are reached through the `*_variables_*`() accessors, [get_index\(\)](#) and [get_axes\(\)](#), not by flat name.

Usage

```
get_metadata(data, fields = NULL)
```

Arguments

data An aniframe or anievent object.

fields Field or category names. A field the object does not carry gives NULL — an [anievent\(\)](#) has no space category, so its spatial fields read as absent rather than as neutral values. A name that is not a metadata field at all is an error.

Value

The metadata associated with the object.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
names(get_metadata(af))

# A single field can be pulled out by name, wherever it lives
get_metadata(af, 'sampling_rate')

# A category name returns the whole category
names(get_metadata(af, 'space'))
```

get_sampling_interval *The interval between consecutive observations*

Description

Derived from the index rather than declared, in the unit the index is in – so a frame indexed by frame number has an interval in frames, and one indexed by seconds has it in seconds.

Usage

```
get_sampling_interval(data)
```

Arguments

data An anipoint object.

Details

Measured per key: identity plus temporal context. The index restarts in each group, so pooling them would measure the restarts rather than the sampling.

Measured again whenever the rows or the index change: by [dplyr::filter\(\)](#), [dplyr::slice\(\)](#), [\[](#), a [dplyr::mutate\(\)](#) of the index or a key, and the other dplyr verbs anicore provides methods for. A frame that keeps every third row of a 30 Hz recording then reports the 10 Hz spacing it now has, and [validate_anipoint\(\)](#) says that its `sampling_rate` no longer matches.

Value

Numeric scalar, or NA when the frame is too short to measure.

See Also

[is_sampling_regular\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 5, n_individuals = 2, n_keypoints = 1)
get_sampling_interval(af)
```

is_anievent	<i>Check if object is an anievent</i>
-------------	---------------------------------------

Description

Check if object is an anievent

Usage

```
is_anievent(x)
```

Arguments

x An object to test.

Value

Logical: TRUE if x inherits from anievent.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_anievent(af)
```

is_aniframe	<i>Check if object is an aniframe</i>
-------------	---------------------------------------

Description

aniframe is the abstract parent of every animovement frame class, so this tests for the whole family: an [anipoint](#), an [anievent](#), and any subclass built on them all pass. Use [is_anipoint\(\)](#) to test for the position grain specifically.

Usage

```
is_aniframe(x)
```

Arguments

x	An object to test
---	-------------------

Value

Logical: TRUE if x inherits from aniframe

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_aniframe(af)

# A plain data frame is not one
is_aniframe(data.frame(x = 1))
```

is_anijoint	<i>Test whether an object is an anijoint</i>
-------------	--

Description

Test whether an object is an anijoint

Usage

```
is_anijoint(x)

ensure_is_anijoint(x)
```

Arguments

x	An object.
---	------------

Value

is_anijoint(): logical. ensure_is_anijoint(): errors if not.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1) |>
  set_structure(example_structure())
is_anijoint(as_anijoint(af))
```

is_anipoint	<i>Check if object is an anipoint</i>
-------------	---------------------------------------

Description

An anipoint is the position-grain frame: one row per point per timepoint, with coordinate columns. Use [is_aniframe\(\)](#) to test for the whole animovement frame family instead.

Usage

```
is_anipoint(x)
```

Arguments

x An object to test

Value

Logical: TRUE if x inherits from anipoint

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_anipoint(af)

# A plain data frame is not one
is_anipoint(data.frame(x = 1))
```

is_anisegment	<i>Test whether an object is an anisegment</i>
---------------	--

Description

Test whether an object is an anisegment

Usage

```
is_anisegment(x)

ensure_is_anisegment(x)
```

Arguments

x An object.

Value

is_anisegment(): logical. ensure_is_anisegment(): errors if not.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1) |>
  set_structure(example_structure())
is_anisegment(as_anisegment(af))
```

is_anistructure	<i>Test whether an object is an anistructure</i>
-----------------	--

Description

Test whether an object is an anistructure

Usage

```
is_anistructure(x)

ensure_is_anistructure(x)
```

Arguments

x An object.

Value

is_anistructure(): logical. ensure_is_anistructure(): errors if not.

Examples

```
is_anistructure(example_structure())
```

is_cartesian	<i>Test whether an anipoint uses a Cartesian coordinate system</i>
--------------	--

Description

Returns TRUE if the data frame satisfies *any* of the 1-D, 2-D or 3-D Cartesian checks.

Usage

```
is_cartesian(data)
```

Arguments

data An anipoint.

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_cartesian(af)
```

is_cartesian_1d	<i>Test for a 1-D Cartesian coordinate system</i>
-----------------	---

Description

Test for a 1-D Cartesian coordinate system

Usage

```
is_cartesian_1d(data)
```

Arguments

data An anipoint.

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_cartesian_1d(af)
```

is_cartesian_2d	<i>Test for a 2-D Cartesian coordinate system</i>
-----------------	---

Description

Test for a 2-D Cartesian coordinate system

Usage

```
is_cartesian_2d(data)
```

Arguments

data	An anipoint.
------	--------------

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_cartesian_2d(af)
```

is_cartesian_3d	<i>Test for a 3-D Cartesian coordinate system</i>
-----------------	---

Description

Test for a 3-D Cartesian coordinate system

Usage

```
is_cartesian_3d(data)
```

Arguments

data	An anipoint.
------	--------------

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_cartesian_3d(af)
```

is_cylindrical	<i>Test whether an anipoint uses a cylindrical coordinate system</i>
----------------	--

Description

Test whether an anipoint uses a cylindrical coordinate system

Usage

```
is_cylindrical(data)
```

Arguments

data	An anipoint.
------	--------------

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_cylindrical(af)
```

is_polar	<i>Test whether an anipoint uses a polar coordinate system</i>
----------	--

Description

Test whether an anipoint uses a polar coordinate system

Usage

```
is_polar(data)
```

Arguments

data	An anipoint.
------	--------------

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_polar(af)
```

is_sampling_regular	<i>Is the frame regularly sampled?</i>
---------------------	--

Description

Every gap between consecutive observations equal, within tolerance. Computed from the data each time it is asked rather than recorded, because dropping rows changes the answer and a stored logical would go on claiming the old one.

Usage

```
is_sampling_regular(data, tolerance = 1e-06)
```

Arguments

data	An anipoint object.
tolerance	Relative tolerance: a gap counts as equal to the interval when it differs by no more than $\text{tolerance} * \text{interval}$. Timestamps are rarely exactly equal, so comparing them with <code>==</code> says "irregular" for data that is regular to any precision that matters. Raise it for noisy timestamps, lower it to be strict.

Value

TRUE, FALSE, or NA when the frame is too short to tell.

See Also

[get_sampling_interval\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 5, n_individuals = 2, n_keypoints = 1)
is_sampling_regular(af)

# A gap in the recording
irregular <- af |> dplyr::filter(time != 3)
is_sampling_regular(irregular)
```

is_spatial	<i>Test whether the spatial columns match the metadata</i>
------------	--

Description

Returns TRUE when `variables_where` declares at least one column and every column it names is present and numeric.

Usage

```
is_spatial(data)
```

Arguments

`data` An anipoint object.

Details

This is a different question from the one `is_cartesian()` and its siblings answer: those test for the presence of particular column *names* (`x`, `y`, `z`, ...) and never consult the metadata or the column types. A frame that has lost its `x` column still satisfies `is_cartesian_1d()` on the strength of `y` alone, while its `variables_where` still promises both.

Value

Logical scalar.

See Also

`ensure_is_spatial()`, `validate_anipoint()`.

Examples

```
af <- anipoint(time = 1:5, x = 1:5, y = 1:5)
is_spatial(af)

# Dropping a declared column breaks the correspondence
is_spatial(dplyr::select(af, -x))
```

is_spherical	<i>Test whether an anipoint uses a spherical coordinate system</i>
--------------	--

Description

Test whether an anipoint uses a spherical coordinate system

Usage

```
is_spherical(data)
```

Arguments

data An anipoint.

Value

A logical value.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
is_spherical(af)
```

list_default_metadata	<i>Default metadata structure</i>
-----------------------	-----------------------------------

Description

Returns the default metadata tree. Fields are grouped into categories (#118) — recording, time, space, variables, structure — with spec_version at the top level. Access stays flat: a field is read with `get_metadata(data, "sampling_rate")` and written with `set_metadata(data, sampling_rate = 30)` regardless of its category, and a category name returns the whole category.

The categories:

- recording — provenance: source, source_version, source_format, source_sampling_rate, filename. source_sampling_rate is the rate the device recorded at, in Hz, and no processing changes it; see [set_metadata\(\)](#) for how it is filled.
- time — unit_time, sampling_rate (declared: the rate of the data as it is now, which resampling changes), sampling_interval (derived from the index; read it with [get_sampling_interval\(\)](#)), start_datetime.
- space — coordinate_system, reference_frame, handedness, axis_directions, axis_extents, unit_space, unit_angle, and euler_sequence / euler_intrinsic: the Euler convention the data's source uses, for showing and entering a quaternion orientation. The one category a class can lack: an [anievent\(\)](#) has no spatial component, so its metadata simply has no space (#73).

- `variables` — which columns play which role, as a list of roles each holding named slots: `what$keys` (identity), `when$index + when$keys` (temporal; an `anievent` has `when$interval` instead of an index), `where$position` (the axis-role mapping; names are roles, values are columns), `event$state + event$point`. The frame groups by `c(what$keys, when$keys)` and nothing else. These slots are reached through `get_variables()`, `get_index()` and `get_axes()`, never by flat name — `keys` appears under two roles.
- `structure` — named `anistructure()`s, each relating the levels of one variable; see `set_structure()`.
- `spec_version` — named list of semantic version strings, one per class, versioning the full data contract independently of the package version.

Usage

```
list_default_metadata(class = c("anipoint", "anievent"))
```

Arguments

<code>class</code>	Which class's tree to return: "anipoint" (the default) or "anievent". An <code>anievent</code> 's tree has no space category, and its <code>when</code> role carries <code>interval = c("start", "stop")</code> instead of an index.
--------------------	--

Value

A named list: the categories above plus `spec_version`.

See Also

`set_metadata()`, `get_metadata()`

Examples

```
names(list_default_metadata())
names(list_default_metadata("anievent"))
```

metadata_json

Read and write a frame's metadata as JSON

Description

`get_metadata_json()` gives a frame's metadata as JSON, which any language can read. `set_metadata_json()` restores metadata from that JSON, so that the round trip gives back exactly what `get_metadata()` gave.

This is how the metadata travels outside R, for instance in the key-value metadata of a Parquet file written by `aniread`.

Usage

```
get_metadata_json(data)

set_metadata_json(data, json)
```

Arguments

<code>data</code>	An aniframe. For <code>set_metadata_json()</code> , a frame of the class the metadata belongs to: its class decides which categories and slots are valid.
<code>json</code>	A JSON string from <code>get_metadata_json()</code> . Or that JSON already parsed with <code>jsonlite::fromJSON(simplifyVector = FALSE)</code> , for when it is embedded in a larger document.

Value

`get_metadata_json()`: a JSON string. `set_metadata_json()`: data with the metadata from json.

Layout

The JSON has the categories of `list_default_metadata()` as objects, in the same order, with `spec_version` saying which version of the layout it follows:

- A field with a fixed set of values (`unit_time`, `unit_space`, `coordinate_system`, ...) is its value as a string.
- NA is null. `source_sampling_rate` can also be the string "NaN": a device with no fixed rate, which JSON has no number for.
- `start_datetime` is an ISO 8601 time in UTC, to the microsecond, with its time zone beside it as `start_timezone` (an IANA name, or "" for the session's own).
- `axis_directions` and `axis_extents` are objects from axis role to value.
- In variables, each role is an object of slots. A slot that maps axis roles to columns (`where$position`) is an object; one that lists columns (`what$keys`) is an array, even with one column.
- `structure` is an object of structures. Each has points (an array), segments and joints (arrays of row objects), and root, source, citation, license and variable.

Numbers are written with as few digits as read back exactly. Inf and -Inf, which JSON also has no number for, are strings like "NaN".

See Also

[get_metadata\(\)](#), [set_metadata\(\)](#)

Examples

```
x <- example_anipoint(n_obs = 2, n_individuals = 1)
json <- get_metadata_json(x)
json

y <- set_metadata_json(x, json)
identical(get_metadata(y), get_metadata(x))
```

```
print.aniframe_metadata
```

Print the metadata of a frame

Description

Prints the metadata of an [anipoint\(\)](#) or [anievent\(\)](#) one category at a time, one name: value line per field, wrapped to the console width. Values carry their units where the metadata declares them, such as `sampling_rate: 30 Hz`. Fields that are not set are left out and named together on a closing line. The rate the device recorded at is shown with the current rate, and only when the two differ, as in `sampling_rate: 50 Hz (recorded at 200 Hz)`.

`all = TRUE` prints every field, including those not set, the values each factor field allows, and `spec_version`. `source_sampling_rate` then has a line of its own.

Usage

```
## S3 method for class 'aniframe_metadata'
print(x, all = FALSE, ...)
```

Arguments

<code>x</code>	An <code>aniframe_metadata</code> object, as returned by get_metadata() .
<code>all</code>	Whether to print every field (default <code>FALSE</code>): those not set, the allowed values of factor fields, and <code>spec_version</code> .
<code>...</code>	Unused.

Value

`x`, invisibly.

Examples

```
md <- get_metadata(set_metadata(example_anipoint(), sampling_rate = 30))
md
print(md, all = TRUE)
```

```
print.anisttructure
```

Print a structure

Description

Prints the counts of a structure's points, segments and joints, then each list wrapped to the console width. A segment prints as `from - to` when its name is the default `from-to`, and as `name: from - to` when it has a name of its own; joints likewise, over their two segments.

Like a tibble's rows, a list of more than 20 prints its first 10 and how many more there are. `n` sets how many to print; `n = Inf` prints all.

Usage

```
## S3 method for class 'anistructure'
print(x, n = NULL, ...)

## S3 method for class 'anistructure'
format(x, n = NULL, width = cli::console_width(), ...)
```

Arguments

x	An anistructure() .
n	How many points, segments and joints to print. NULL (the default) prints a list in full up to 20 entries, and the first 10 of a longer one.
...	Unused.
width	Width to wrap the lists to (default the console width).

Value

print(): x, invisibly. format(): a character vector of lines.

Examples

```
s <- example_structure()
s
print(s, n = 3)
```

rad_to_deg	<i>Convert radians to degrees</i>
------------	-----------------------------------

Description

Convert radians to degrees

Usage

```
rad_to_deg(x)
```

Arguments

x	Numeric vector of angles (radians).
---	-------------------------------------

Value

Numeric vector of angles expressed in degrees.

See Also

Other angle utilities: [angle_to_rad\(\)](#), [deg_to_rad\(\)](#), [unwrap_angle\(\)](#), [wrap_angle\(\)](#)

Examples

```
rad_to_deg(pi)
rad_to_deg(c(0, pi / 2, pi))
```

reflect_axis

Turn an axis over

Description

Reflects the column carrying an axis role and flips its declared direction, so the data describes the same scene with the axis pointing the other way — for example converting image coordinates, where y runs down, to a y that runs up.

An axis runs from zero to its extent, so turning it over gives $\text{new} = \text{extent} - \text{old}$. An axis with no declared `axis_extents` is centred on its origin, and turning it over negates it. On a frame that stores angles there is no column to reflect, and `phi` and `theta` are recomputed instead.

A declared handedness flips with any linear axis.

Usage

```
reflect_axis(data, axis)
```

Arguments

<code>data</code>	An anipoint object.
<code>axis</code>	An axis role: "x", "y" or "z".

Value

The anipoint, reflected, with its orientation metadata updated.

See Also

[set_axis_directions\(\)](#), [get_handedness\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
af <- set_metadata(af, axis_extents = c(y = 1080))
af <- set_axis_directions(af, c(x = "right", y = "down"))

af <- reflect_axis(af, "y")
get_axis_directions(af)
```

set_axis_directions	<i>Say which way an axis points</i>
---------------------	-------------------------------------

Description

Records the direction of one or more axes, keyed by axis role. Roles not named keep the direction they had. This only declares: the values are left alone. To turn an axis over and keep the data describing the same scene, use [reflect_axis\(\)](#).

Usage

```
set_axis_directions(data, directions)
```

Arguments

data	An anipoint object.
directions	Named character vector, axis role to direction — one of right, left, up, down, back or forward. NA clears an axis.

Details

Directions are read from where the recording was made: right/left across the view, up/down within it, back/forward toward and away from the viewer. No two axes may point along the same pair. Three declared directions fix the handedness, which is recorded too.

Value

The anipoint, with the new directions recorded.

See Also

[get_axis_directions\(\)](#), [reflect_axis\(\)](#), [get_angle_direction\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
af <- set_axis_directions(af, c(x = "right", y = "down"))
get_axis_directions(af)
get_angle_direction(af)
```

set_index

*Declare which column an anipoint is indexed by***Description**

Makes column the index, the when\$index slot, and re-sorts the frame. A column that was a when key stops being one. unit declares the unit of the new index in the same call; without it, the frame goes on declaring the unit of the old index.

The new index must be numeric, with no missing values, and must increase within each group of keys (identity plus temporal context) in the order of the old index, so the rows keep their order: values that go backwards have been matched to the wrong rows. Date-times are refused: the index holds the time since the start, as numbers, and the start goes in start_datetime, so absolute time is start_datetime plus the index.

Usage

```
set_index(data, column, unit = NULL)
```

Arguments

data	An anipoint object.
column	Length-one character vector naming the index column. It must exist in data and be numeric.
unit	The unit of the new index, one of the levels of unit_time in list_default_metadata() . NULL, the default, leaves unit_time as it is.

Value

data, re-indexed and restructured.

The old index

The previous index becomes an ordinary, undeclared column rather than a key, as grouping by it would put each row in a group of its own.

When it counted frames (unit_time is "frame"), it is renamed frame. Frame numbers recorded with the data are data, not something to compute again from a rate, and [convert_unit_time\(\)](#) makes the frame column the index again when converting to "frame". An old index already named frame keeps its name, and so does one replaced by a new index named frame. If the frame has another column named frame, set_index() refuses rather than overwrite either; rename one first.

Indexing by recorded timestamps

Timestamps recorded alongside the data, such as a camera's log of when each frame was taken, are added as a column and made the index with their unit:

```
data |>
  dplyr::mutate(timestamp = stamps[time + 1]) |>
  set_index("timestamp", unit = "s")
```

Here the log has one entry per frame and frames count from 0, so the timestamp of a row is matched by its frame number. Matching by position, assigning the log in row order, lines up only when every group is complete and sorted: an anipoint has one row per time per key, and a keypoint can be missing from some frames. Bringing a log, or any other data, into a frame is a join question, tracked in [anicore#1](#).

The frame numbers move to a column named `frame`, and a declared `sampling_rate` is kept as the nominal rate. `sampling_interval` is measured from the timestamps, and `validate_anipoint()` allows the two to differ by 1%. Overwriting time with `dplyr::mutate()` instead would lose the frame numbers and leave the frame saying the index counts frames when it holds seconds.

See Also

`get_index()`, `convert_unit_time()` to rescale the index into another unit.

Examples

```
df <- data.frame(frame = 1:3, individual = "a", x = c(1, 2, 3), y = c(0, 1, 0))
af <- as_anipoint(df, index = "frame")
get_index(af)

# A camera log, one timestamp per frame, frames counted from 0
af <- as_anipoint(data.frame(
  individual = "a",
  keypoint = rep(c("head", "tail"), each = 4),
  time = rep(0:3, 2),
  x = 1:8,
  y = 1:8
))
stamps <- c(0, 0.0332, 0.0668, 0.1001)
logged <- af |>
  dplyr::mutate(timestamp = stamps[time + 1]) |>
  set_index("timestamp", unit = "s")
logged

# The frame numbers are kept, and become the index again
get_index(convert_unit_time(logged, "frame"))
```

Description

Sets or updates metadata for an aniframe or anievent object. Metadata can be provided either as named arguments or as a list. If the object already has metadata, the new values will be merged with existing values, with new values taking precedence.

Fields are written by their own name wherever they live in the category tree (see [list_default_metadata\(\)](#)): `set_metadata(data, sampling_rate = 30)` lands in time, `set_metadata(data, handedness = "left")` in space. Categories themselves are not writable, and the variable declaration goes through its dedicated setters.

Character values for factor fields will be automatically converted to factors if they match allowed levels.

Usage

```
set_metadata(data, ..., metadata = NULL)
```

Arguments

<code>data</code>	An aniframe or anievent object.
<code>...</code>	Named metadata values (e.g., <code>sampling_rate = 30</code> , <code>source = "sleep"</code>)
<code>metadata</code>	Alternatively, a named list of metadata. Cannot be used simultaneously with <code>...</code>

Value

The object with updated metadata.

The recorded and the current sampling rate

A frame keeps two rates apart. `source_sampling_rate`, with the other provenance in recording, is the rate the device recorded at, and no processing changes it. `sampling_rate`, in time, is the rate of the data as it is now: the rate filters and conversions compute with, and the one resampling changes.

The first declaration of a rate fills both. Setting `sampling_rate` on a frame that has neither rate yet also sets `source_sampling_rate`, since that is the moment the camera's rate is declared; this holds for `as_anipoint(metadata = )` too. After that `sampling_rate` changes on its own, so correcting a wrong camera rate means setting `source_sampling_rate` as well, in the same call.

A device with no fixed rate, such as an event-driven sensor whose readings a reader integrates into windows, is declared with `source_sampling_rate = NaN`: "no fixed rate", as distinct from NA, "not declared yet". The first declaration then leaves it alone, and `is.na()` is still TRUE for code that only asks whether there is a recorded rate to use.

See Also

[get_metadata\(\)](#), [list_default_metadata\(\)](#)

Examples

```
data <- example_anipoint()

# Set metadata using named arguments
data <- set_metadata(data, sampling_rate = 30, source = "sleap")

# Set metadata using a list
md <- list(sampling_rate = 30, source = "sleap")
data <- set_metadata(data, metadata = md)

# The first declared rate is also recorded as the device's
af <- set_metadata(example_anipoint(), sampling_rate = 200)
get_metadata(af, "source_sampling_rate")

# Later rates are the data's own, as after resampling
af <- set_metadata(af, sampling_rate = 50)
get_metadata(af, c("sampling_rate", "source_sampling_rate"))

# A sensor with no fixed rate
trackball <- example_anipoint() |>
  set_metadata(source_sampling_rate = NaN) |>
  set_metadata(sampling_rate = 100)
get_metadata(trackball, "source_sampling_rate")
```

structures

Attach, read and remove the structures of a frame

Description

[Experimental]

A frame can carry any number of named `anistructure()`s, each spanning the levels of one identity or temporal variable. Several may span the same variable and overlap: a football frame can hold team, defence and left_flank over individual alongside a skeleton over keypoint. A structure's points are its members; levels it does not list are outside it.

- `set_structure()` attaches a structure under name, replacing one of the same name.
- `get_structure()` returns all structures as a named list, or one.
- `remove_structure()` drops one.

Points not found in the data are kept with a warning, in case they are recorded in another file.

Usage

```
set_structure(data, structure, variable = "keypoint", name = variable)

get_structure(data, name = NULL)

remove_structure(data, name)
```

Arguments

data	An anipoint.
structure	An anisttructure() .
variable	The what or when key whose levels the points are.
name	Name to store the structure under. Defaults to variable.

Value

`set_structure()` and `remove_structure()`: data, with the structures updated. `get_structure()`: a named list of structures, or one structure.

See Also

[anisttructure\(\)](#)

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 2)
af <- set_structure(af, example_structure())
get_structure(af, "keypoint")

# Several structures over the same variable
af <- af |>
  set_structure(anisttructure(points = c("1", "2")), "individual", "pair") |>
  set_structure(anisttructure(points = "1"), "individual", "focal")
names(get_structure(af))
```

to_anievent	<i>Encode per-frame data into an anievent</i>
-------------	---

Description

Run-length-encodes per-frame state and point variables into the long-format [anievent\(\)](#). Works on a data frame (with bare-name selection of the event columns and explicit time / identity) or on an [anipoint\(\)](#) (where everything is read from metadata).

Usage

```
to_anievent(data, ...)

## S3 method for class 'anievent'
to_anievent(data, ...)

## S3 method for class 'data.frame'
to_anievent(
  data,
```

```

    time,
    state = NULL,
    point = NULL,
    variables_what = NULL,
    variables_when = NULL,
    metadata = list(),
    ...
)

## S3 method for class 'anipoint'
to_anievent(
  data,
  variables_what = NULL,
  variables_when = NULL,
  metadata = list(),
  ...
)
```

Arguments

data	A data frame or an anipoint() .
...	Passed to methods.
time	For data-frame input, the column holding per-frame times. Bare name (tidyselect). Required.
state	For data-frame input, columns to run-length-encode as state bouts. Bare names (tidyselect). Logical columns produce bouts on TRUE-runs, labelled by the column name; factor or character columns produce one bout per contiguous non-NA run of the same value.
point	For data-frame input, columns to encode as point bouts. Bare names (tidyselect). Logical columns produce one point bout per TRUE frame, labelled by the column name; factor or character columns produce one bout per non-NA frame.
variables_what	For data-frame input, identity columns (e.g. individual). Bare names (tidyselect). Bouts are isolated per identity group.
variables_when	For data-frame input, additional temporal- grouping columns (e.g. observation, session, trial). Bare names (tidyselect). Like identity, these isolate bouts.
metadata	Optional list of metadata attached to the result. For an anipoint input, fields like unit_time and sampling_rate are propagated automatically; metadata overrides those.

Details

Distinct from [as_anievent\(\)](#): that one is a strict cast — the input must already be in canonical anievent shape (one row per bout, with channel / type / label / start / stop). [to_anievent\(\)](#) is the encoding verb that *produces* that shape from per-frame data.

Value

An [anievent\(\)](#).

Examples

```
df <- data.frame(
  individual = 1L,
  time = 1:8,
  behaviour = factor(c("REM", "REM", "REM", "wake", "wake", "REM", "REM", NA)),
  woke_up = c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE),
  call = c(NA, "alarm", NA, NA, NA, NA, NA, NA)
)
to_anievent(
  df,
  time = time,
  state = c(behaviour, woke_up),
  point = call,
  variables_what = individual
)
```

unwrap_angle

Remove wrapping from a sequence of angles

Description

Reverses the discontinuity introduced by wrapping, by accumulating the shortest step between successive angles. A heading that crosses the end of its range, π in the signed range [wrap_angle\(\)](#) gives, therefore continues to increase rather than jumping back to $-\pi$, which is what makes it differentiable. NA values are preserved in place.

Usage

```
unwrap_angle(x)
```

Arguments

x A numeric vector of angles, in radians, in any range.

Value

A numeric vector the same length as **x**, without wrapping discontinuities. It starts at the first non-missing angle of **x** and is not confined to any range.

See Also

Other angle utilities: [angle_to_rad\(\)](#), [deg_to_rad\(\)](#), [rad_to_deg\(\)](#), [wrap_angle\(\)](#)

Examples

```
# A heading turning steadily past a full circle, wrapped to (-pi, pi]
wrapped <- wrap_angle(seq(0, 3 * pi, length.out = 7))
wrapped

# Unwrapping restores the steady progression
unwrap_angle(wrapped)
```

validate_anievent	<i>Validate an anievent</i>
-------------------	-----------------------------

Description

Re-runs the structural invariants of an anievent object on demand:

- required columns (channel, type, label, start, stop) are present with the expected types — hard errors;
- type is a factor with levels c("state", "point") — hard error;
- stop >= start for every row — hard error;
- modifiers, if present, is a list-column whose cells are character vectors — hard error;
- within each (identity + temporal-grouping) group, two bouts of the same channel should not overlap — **warning** only. Overlapping bouts within a channel are permitted on the anievent side; this is the form the data takes from coding tools that allow non-mutually-exclusive coding (e.g. BORIS without strict ethogram).

Usage

```
validate_anievent(data)
```

Arguments

data An anievent object.

Value

The input data, invisibly.

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
try(validate_anievent(af))
```

validate_anipoint	<i>Validate an anipoint</i>
-------------------	-----------------------------

Description

Re-checks, on demand, that an anipoint's metadata still describes the frame it is attached to. The two drift apart silently under ordinary dplyr work: `dplyr::select()` drops a column without touching the metadata that names it, and assignment can change a column's type. The invariants are therefore checked rather than assumed:

Usage

```
validate_anipoint(data, rate_tolerance = 0.01)
```

Arguments

<code>data</code>	An anipoint object.
<code>rate_tolerance</code>	How far the declared <code>sampling_rate</code> may be from the rate the index is spaced at before it warns, relative to the declared rate. The default, <code>0.01</code> , lets real timestamps through: a camera log that averages 30.11 Hz agrees with a declared 30 Hz. Lower it for a strict check, such as <code>1e-6</code> for an index computed from the rate.

Details

- the index column is present and numeric — hard error;
- every declared column is present in the data — hard error;
- every position column is numeric — hard error;
- `coordinate_system` agrees with the axis roles — **warning** only. The frame is still usable, and the field is derived rather than declared, so it can be refreshed;
- identity, temporal context and the index together name one observation per row — **warning** only (#49);
- a declared `sampling_rate` agrees with the spacing of the index, to within `rate_tolerance` — **warning** only (#114). The spacing is the median gap, so a few dropped frames do not count against it.

Value

The input data, invisibly.

See Also

`ensure_is_spatial()` for the spatial subset of these checks, which is the part downstream filters need on every call; `validate_anievent()` for the anievent equivalent.

Examples

```
af <- anipoint(time = 1:5, x = 1:5, y = 1:5)
validate_anipoint(af)
```

validate_anisttructure *Check that a structure is internally consistent*

Description

[Experimental]

Checks the rules a structure has to satisfy:

- points are unique and not NA;
- every segment joins two known, different points, and any expected length is positive;
- every joint pairs two different, defined segments, about an axis that is "x", "y", "z" or a segment name;
- a joint's min is not above its max, and its rest lies between them;
- root is one of the points, or NA.

`anisttructure()` and `set_structure()` call it, so a structure built there is already valid; call it yourself after editing a structure's parts by hand.

Usage

```
validate_anisttructure(x)
```

Arguments

x An anisttructure.

Value

x, invisibly; errors naming the first problem otherwise.

Examples

```
validate_anisttructure(example_structure())
```

variables	<i>Read and declare which columns carry identity, time, position and events</i>
-----------	---

Description

The variables metadata category says which columns play which role. Each role is a list of named slots:

- what: keys, the identity columns.
- when: keys, the temporal context (session, trial), plus index on an anipoint or interval (start, stop) on an anievent.
- where: position, axis role to column, and optionally orientation: yaw in 2D or a unit quaternion qw, qx, qy, qz in 3D (anipoint only).
- event: state and point, the per-frame event columns (anipoint only).

The frame is grouped by the keys of what and when; see [get_keys\(\)](#).

Declaring restructures the frame to match — columns are retyped, reordered and regrouped, and coordinate_system is re-derived — so the metadata and the frame cannot drift apart. A column must exist before it is declared.

- set_variables() replaces the slots it is given and leaves the rest.
- add_variables() appends to them.
- remove_variables() drops columns from them.

Each role argument takes a named list of slots, or a character vector for the role's main slot: keys for what and when, position for where. In remove_variables() a character vector drops the columns from every slot of the role.

Usage

```
get_variables(data, role = NULL, slot = NULL)
```

```
set_variables(data, what = NULL, when = NULL, where = NULL, event = NULL)
```

```
add_variables(data, what = NULL, when = NULL, where = NULL, event = NULL)
```

```
remove_variables(data, what = NULL, when = NULL, where = NULL, event = NULL)
```

Arguments

data	An aniframe.
role	Role to read, or NULL for the whole category.
slot	Slot within role, or NULL for the union of its slots.
what, when, where, event	A named list of slots, or a character vector for the role's main slot.

Value

`get_variables()`: the whole category; the columns of a role, unnamed; or one slot as stored. The setters: `data`, `restructured`.

See Also

`get_keys()`, `get_index()`, `set_index()`, `get_axes()`

Examples

```
af <- example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 1)
get_variables(af, "what")
get_variables(af, "where", "position")

# Declaring an identity column groups the frame by it
af |>
  dplyr::mutate(id = "a") |>
  add_variables(what = "id") |>
  dplyr::group_vars()

# Axis roles map to columns of any name
df <- data.frame(time = 1:3, u = c(1, 2, 3), v = c(0, 1, 0))
as_anipoint(df, variables_where = c("u", "v")) |>
  set_variables(where = c(x = "u", y = "v")) |>
  get_metadata("coordinate_system")
```

wrap_angle

Constrain angles to a standard range

Description

Wraps a vector of angles to a standard interval using modulo arithmetic. By default that is the signed range $(-\pi, \pi]$, which the suite uses for every direction; see the section below.

Usage

```
wrap_angle(x, modulo = c("pi", "2pi", "asis"))
```

Arguments

x	A numeric vector of angles, in radians.
modulo	A character string (default "pi") giving the target range: "pi" Wrap to $(-\pi, \pi]$. "2pi" Wrap to $[0, 2\pi]$. "asis" Return unchanged.

Value

A numeric vector the same length as `x`, wrapped to the chosen range.

The range of a direction

Directions throughout the animovement suite are signed, in $(-\pi, \pi]$ ($(-180, 180]$ in degrees). It is the range `atan2()` returns, so per-row directions such as a course or a heading come out in it already, and the circular summaries `circ_mean()` and `circ_median()` return it too, so a summary can be compared with the values it summarises. 0 points along x , and the sign says which side of x a direction lies on: positive toward y , negative away from it. Which way that turns on screen is the frame's angle direction, `get_angle_direction()`.

The range does not change what a direction means: $-\pi / 2$ and $3 * \pi / 2$ are the same direction, and every `circ_*`() function treats them alike. To report directions in $[0, 2\pi)$ instead, wrap at the end with `wrap_angle(x, "2pi")`. Differences (`circ_difference()`) are signed in $(-\pi, \pi]$ as well, while unwrapped and cumulative angles (`unwrap_angle()`) are not confined to any range. The **orientation article** covers the convention alongside the frame's axis directions.

See Also

Other angle utilities: `angle_to_rad()`, `deg_to_rad()`, `rad_to_deg()`, `unwrap_angle()`

Examples

```
angles <- c(-pi, 0, pi, 3 * pi / 2, 2 * pi, 3 * pi)

# The signed range, which directions use throughout the suite
wrap_angle(angles)

# The same angles on  $[0, 2\pi)$ 
wrap_angle(angles, "2pi")

# "asis" is a no-op, useful when the range is chosen by a caller
wrap_angle(angles, "asis")
```

Index

* angle utilities

- angle_to_rad, 4
- deg_to_rad, 24
- rad_to_deg, 52
- unwrap_angle, 61
- wrap_angle, 66

* circular statistics

- circ_difference, 15
- circ_mad, 15
- circ_mean, 16
- circ_median, 17
- circ_sd, 18
- circ_successive_difference, 18

- add_variables(variables), 65
- angle_between, 3
- angle_between(), 11, 12
- angle_from_rad(angle_to_rad), 4
- angle_to_rad, 4
- angle_to_rad(), 24, 52, 61, 67
- anievent, 5, 40
- anievent(), 36, 38, 48, 51, 59, 60
- anipoint, 6, 40
- anipoint(), 5, 51, 59, 60
- anistructure, 8
- anistructure(), 12, 14, 49, 52, 58, 59, 64
- as_anievent, 10
- as_anievent(), 60
- as_anijoint, 11
- as_anipoint, 12
- as_anipoint(), 14
- as_anisegment, 14
- as_anisegment(), 12, 13
- atan2(), 67

- base::diff(), 18

- circ_difference, 15
- circ_difference(), 16–19, 67
- circ_mad, 15

- circ_mad(), 15–19
- circ_mean, 16
- circ_mean(), 15–19, 67
- circ_median, 17
- circ_median(), 15, 16, 18, 19, 67
- circ_sd, 18
- circ_sd(), 15–17, 19
- circ_successive_difference, 18
- circ_successive_difference(), 15–18
- convert_inf_to_na, 19
- convert_nan_to_na, 20
- convert_nan_to_na(), 19
- convert_unit_angle, 20
- convert_unit_angle(), 4, 21
- convert_unit_space, 21
- convert_unit_time, 22
- convert_unit_time(), 55, 56
- deg_to_rad, 24
- deg_to_rad(), 5, 52, 61, 67
- dplyr::filter(), 38
- dplyr::mutate(), 4, 18, 38, 56
- dplyr::select(), 63
- dplyr::slice(), 38
- ensure_is_anievent, 24
- ensure_is_aniframe, 25
- ensure_is_anijoint(is_anijoint), 40
- ensure_is_anipoint, 25
- ensure_is_anipoint(), 25
- ensure_is_anisegment(is_anisegment), 42
- ensure_is_anistructure
(is_anistructure), 42
- ensure_is_cartesian, 26
- ensure_is_cartesian_1d, 27
- ensure_is_cartesian_2d, 27
- ensure_is_cartesian_3d, 28
- ensure_is_cylindrical, 28
- ensure_is_polar, 29
- ensure_is_spatial, 29

ensure_is_spatial(), 47, 63
 ensure_is_spherical, 30
 example_anipoint, 31
 example_anipoint(), 32
 example_structure, 32
 example_structure(), 9

 format.anisttructure
 (print.anisttructure), 51

 get_angle_direction, 32
 get_angle_direction(), 11, 36, 54, 67
 get_axes, 33
 get_axes(), 35, 37, 49, 66
 get_axis_directions, 34
 get_axis_directions(), 32, 33, 36, 54
 get_coordinate_system, 35
 get_handedness, 35
 get_handedness(), 33, 53
 get_index, 36
 get_index(), 37, 49, 56, 66
 get_keys, 37
 get_keys(), 65, 66
 get_metadata, 37
 get_metadata(), 49–51, 57
 get_metadata_json (metadata_json), 49
 get_sampling_interval, 38
 get_sampling_interval(), 46, 48
 get_structure (structures), 58
 get_variables (variables), 65
 get_variables(), 36, 37, 49

 is_anievent, 39
 is_aniframe, 40
 is_aniframe(), 25, 41
 is_anijoint, 40
 is_anipoint, 41
 is_anipoint(), 40
 is_anisegment, 42
 is_anisttructure, 42
 is_cartesian, 43
 is_cartesian(), 35, 47
 is_cartesian_1d, 43
 is_cartesian_2d, 44
 is_cartesian_3d, 44
 is_cylindrical, 45
 is_polar, 45
 is_polar(), 35
 is_sampling_regular, 46

 is_sampling_regular(), 39
 is_spatial, 47
 is_spatial(), 29, 30
 is_spherical, 48

 list_default_metadata, 48
 list_default_metadata(), 21, 23, 37, 50, 55, 57
 list_recognised_variables_what(), 7, 13

 metadata_json, 49

 print.aniframe_metadata, 51
 print.anisttructure, 51

 rad_to_deg, 52
 rad_to_deg(), 5, 24, 61, 67
 reflect_axis, 53
 reflect_axis(), 34, 54
 remove_structure (structures), 58
 remove_variables (variables), 65

 set_axis_directions, 54
 set_axis_directions(), 34, 53
 set_index, 55
 set_index(), 23, 36, 66
 set_metadata, 56
 set_metadata(), 22, 48–50
 set_metadata_json (metadata_json), 49
 set_structure (structures), 58
 set_structure(), 8, 9, 14, 49, 64
 set_variables (variables), 65
 set_variables(), 33, 35
 structures, 58

 to_anievent, 59
 to_anievent(), 11

 unwrap_angle, 61
 unwrap_angle(), 5, 24, 52, 67

 validate_anievent, 62
 validate_anievent(), 63
 validate_anipoint, 63
 validate_anipoint(), 23, 30, 38, 47, 56
 validate_anisttructure, 64
 variables, 65

 wrap_angle, 66
 wrap_angle(), 5, 16, 17, 24, 52, 61