# Package 'actibase'

September 17, 2026

**Type** Package

**Title** Baseline Functions for Actigraphy and Activity Processing and Analysis

**Version** 0.6.0

**Description** Provides baseline functions for actigraphy and activity data. This package is intended to be extended by downstream overlays such as 'actiread', 'actimetrics', and 'stepcount'.

**License** GPL-3

**Depends** R (>= 4.1.0)

**Suggests** testthat (>= 3.0.0), utils, covr, knitr, readr

**Encoding** UTF-8

**LazyData** true

**Imports** magrittr, dplyr, lubridate, hms, tidyr, janitor, assertthat, vctrs

**URL** https://github.com/jhuwit/actibase, https://jhuwit.github.io/actibase/

**BugReports** https://github.com/jhuwit/actibase/issues

**Config/roxygen2/version** 8.0.0

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** John Muschelli [aut, cre] (ORCID: <https://orcid.org/0000-0001-6469-1750>)

**Maintainer** John Muschelli <muschellij2@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-09-17 16:00:02 UTC

# Contents



---

`acti_day_index` *Calculate a Baseline-Relative Activity-Day Index*

---

## Description

Calculate a Baseline-Relative Activity-Day Index

## Usage

```
acti_day_index(
  time,
  day_start = hms::hms(0),
  origin = NULL,
  start = 1L,
  timezone = NULL
)
```

### Arguments

| | |
|---|---|
| time | A ‘POSIXt‘ vector. |
| day_start | An ‘hms‘ or ‘HH:MM[:SS]‘ day boundary. |
| origin | Optional ‘Date‘ or ‘POSIXt‘ baseline; by default the first day. |
| start | The index assigned to the baseline day, either ‘0‘ or ‘1‘. |
| timezone | Optional Olson timezone used to define the day boundary. |

### Value

An integer vector.

### Note

A stable recording-relative day number supports faceting and alignment across participants even when their calendar start dates differ.

### Examples

```
time = as.POSIXct(c("2020-01-01 03:00:00", "2020-01-01 05:00:00"), tz = "UTC")
acti_day_index(time, day_start = "04:00")
```

---

## acti_day_start *Find the Start of an Activity Day*

---

### Description

This is a custom-day-boundary alternative to [lubridate::floor_date()] with ‘unit = "day"‘. Use ‘floor_date()‘ when days begin at midnight; use ‘acti_day_start()‘ when a recording day begins at a different time, such as 04:00.

### Usage

```
acti_day_start(time, day_start = hms::hms(0), timezone = NULL)
```

### Arguments

| | |
|---|---|
| time | A ‘POSIXt‘ vector. |
| day_start | An ‘hms‘ or ‘HH:MM[:SS]‘ day boundary. |
| timezone | Optional Olson timezone used to define the day boundary. |

### Value

A ‘POSIXct‘ vector.

## Note

Activity studies often use a non-midnight boundary (for example, 04:00) so nocturnal behavior is assigned to the preceding activity day.

## Examples

```
time = as.POSIXct(c("2020-01-01 03:00:00", "2020-01-01 05:00:00"), tz = "UTC")

# With a midnight boundary, this agrees with floor_date().
acti_day_start(time)
lubridate::floor_date(time, unit = "day")

# With a 04:00 boundary, 03:00 belongs to the preceding activity day.
acti_day_start(time, day_start = "04:00")
```

---

# acti_fill_zeros *Fill in Zeros*

---

## Description

Fill in Zeros

## Usage

```
acti_fill_zeros(data)

acti_fill_zeroes(data)

acti_remove_leading_zeros(data)

acti_remove_leading_zeroes(data)
```

## Arguments

| | |
|---|---|
| data | a data frame containing columns time, X, Y, Z |

## Value

the modified data frame with zeros replaced by NA

## Examples

```
acti_fill_zeros(acti_raw_data)
acti_fill_zeroes(acti_raw_data)
```

## acti_hms_to_minute *Convert an HMS Time to a Minute Index*

### Description

Convert an HMS Time to a Minute Index

### Usage

```
acti_hms_to_minute(x, start = 1L)
```

### Arguments

| | |
|---|---|
| x | An ‘hms‘, ‘POSIXt‘, or ‘HH:MM[:SS]‘ character vector. |
| start | The index assigned to midnight, either ‘0‘ or ‘1‘. |

### Value

An integer minute-of-day index.

### Note

This is a strict ‘hms‘ entry point for callers that have already separated time of day from date and want the standard activity-day index.

### Examples

```
acti_hms_to_minute(hms::hms(hours = c(0, 23), minutes = c(0, 59)))
```

## acti_minute_to_hms *Convert a Minute Index to an HMS Time*

### Description

Convert a Minute Index to an HMS Time

### Usage

```
acti_minute_to_hms(minute, start = 1L)
```

### Arguments

| | |
|---|---|
| minute | Whole-minute positions in a day. |
| start | The index assigned to midnight, either ‘0‘ or ‘1‘. |

## Value

An ‘hms‘ vector.

## Note

This is the inverse of [acti_time_to_minute()] for readable axis labels and for joining minute-indexed results back to a time-of-day value.

## Examples

```
acti_minute_to_hms(c(1, 721, 1440))
acti_minute_to_hms(c(0, 1439), start = 0)
```

---

# acti_raw_data *Example Actigraphy/Activity Raw Data*

---

## Description

Example Actigraphy/Activity Raw Data

## Usage

```
acti_raw_data
```

## Format

A data.frame with the columns

**time** time column

**X** X-axis column

**Y** Y-axis column

**Z** Z-axis column

---

# acti_repeat_time_of_day *Repeat a Time of Day Across a Recording*

---

## Description

Repeat a Time of Day Across a Recording

## Usage

```
acti_repeat_time_of_day(seconds, time, timezone = NULL)
```

### Arguments

| | |
|---|---|
| seconds | Seconds after midnight, or an ‘hms‘/‘HH:MM[:SS]‘ time. |
| time | A ‘POSIXt‘ recording time vector used to determine dates and timezone. |
| timezone | Optional Olson timezone; by default the timezone of ‘time‘. |

### Value

A ‘POSIXct‘ vector with one occurrence per recording date.

### Note

This supplies daily plotting boundaries such as a fixed bedtime or wake time for every calendar date present in a recording.

### Examples

```
time = as.POSIXct(c("2020-01-01 12:00:00", "2020-01-03 12:00:00"), tz = "UTC")
acti_repeat_time_of_day("01:30", time)
```

---

## acti_resample *Resample 3-axial input signal to a specific sampling rate*

---

### Description

Resample 3-axial input signal to a specific sampling rate

### Usage

```
acti_resample(data, sample_rate, method = "linear", ...)

acti_resample_to_time(data, times, method = "linear", ...)
```

### Arguments

| | |
|---|---|
| data | A ‘data.frame‘ with a column for time in ‘POSIXct‘ (usually ‘time‘), and ‘X‘, ‘Y‘, ‘Z‘ |
| sample_rate | sampling frequency, coercible to an integer. This is the sampling rate you’re sampling the data *into*. |
| method | method for interpolation. Options are ‘"linear"/"constant"‘, which uses ‘stats::approx‘, or one of ‘"fmm", "periodic", "natural", "monoH.FC", "hyman"‘, which uses ‘stats::spline‘ |
| ... | additional arguments to pass to [stats::approx()] or [stats::spline] |
| times | a vector of ‘POSIXct‘ date/time values to interpolate the data to |

## Value

A ‘data.frame‘/‘tibble‘ of ‘time‘ and ‘X‘, ‘Y‘, ‘Z‘.

## Examples

```
old = options(digits.secs = 3)
x = acti_raw_data
res = acti_resample(data = x, sample_rate = 80)
res = acti_resample(data = x, sample_rate = 100)
res = acti_resample(data = x, sample_rate = 1)
res = acti_resample_to_time(
  data = x,
  times = lubridate::floor_date(x$time, unit = "1 sec"),
)
res_nat = acti_resample_to_time(
  data = x,
  times = lubridate::floor_date(x$time, unit = "1 sec"),
  method = "natural"
)
options(old)
```

---

# acti_separate_time *Separate Times into Date, Hour, and Minute*

---

## Description

Separate Times into Date, Hour, and Minute

## Usage

```
acti_separate_time(data)

acti_separate_times(data)

acti_create_date(data)

acti_create_hour(data)

acti_create_minute(data)
```

## Arguments

| | |
|---|---|
| data | a ‘data.frame‘ with a ‘time‘ column |

## Value

A ‘data.frame‘ with date, hour, minute, and day columns

## Examples

```
library(actibase)
acti_separate_time(acti_raw_data)
acti_create_date(acti_raw_data)
acti_create_hour(acti_raw_data)
acti_create_minute(acti_raw_data)
```

---

`acti_split_intervals_at_midnight`
*Split Intervals at Midnight*

---

## Description

Split Intervals at Midnight

## Usage

```
acti_split_intervals_at_midnight(start, end, timezone = NULL)
```

## Arguments

| | |
|---|---|
| `start, end` | Equally sized 'POSIXt' vectors. |
| `timezone` | Optional Olson timezone used to define midnight. |

## Value

A data frame with input 'interval' number and split 'start'/'end'.

## Note

Sleep, wear, and activity episodes can span dates; splitting them makes their durations and faceted daily displays unambiguous.

## Examples

```
start = as.POSIXct("2020-01-01 23:30:00", tz = "UTC")
end = as.POSIXct("2020-01-02 00:30:00", tz = "UTC")
acti_split_intervals_at_midnight(start, end)
```

`acti_standardize_data` *Standardize the Accelerometry Data*

## Description

Standardize the Accelerometry Data

## Usage

```
acti_standardize_data(
  data,
  subset_xyz = TRUE,
  colname_time = "time",
  lower_case = FALSE,
  check_xyz = TRUE
)

acti_standardise_data(
  data,
  subset_xyz = TRUE,
  colname_time = "time",
  lower_case = FALSE,
  check_xyz = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | A ‘data.frame‘ with a column for time in ‘POSIXct‘ (usually ‘time‘), and ‘X‘, ‘Y‘, ‘Z‘ |
| `subset_xyz` | should only the ‘time‘ (if available) and ‘XYZ‘ be subset? |
| `colname_time` | column name of header for time |
| `lower_case` | flag to make sure all columns are lower case. |
| `check_xyz` | Check if X/Y/Z is in the data |

## Value

A ‘data.frame‘ with ‘X/Y/Z‘ and a time in ‘time‘ (if available).

## Examples

```
acti_standardize_data(acti_raw_data)
acti_standardise_data(acti_raw_data)
```

## acti_tidy_axes *Tidy axes to a long format*

### Description

Tidy axes to a long format

### Usage

```
acti_tidy_axes(data, colname_time = "time")
```

### Arguments

| | |
|---|---|
| data | An object with columns a time column ‘X‘, ‘Y‘, and ‘Z‘ or an object of class ‘AccData‘ |
| colname_time | column name of header for time |

### Value

A long data set with ‘time‘, ‘axis‘, and ‘value‘

### Examples

```
long = acti_tidy_axes(acti_raw_data)
```

## acti_time_of_day *Extract the Time of Day*

### Description

Extract the Time of Day

### Usage

```
acti_time_of_day(time, timezone = NULL)
```

### Arguments

| | |
|---|---|
| time | A ‘POSIXt‘ vector. |
| timezone | Optional Olson timezone in which to extract time of day. |

### Value

An ‘hms‘ vector.

## Note

This explicitly applies the recording or requested timezone before dropping the date, which prevents a timestamp from being assigned to the wrong activity-day minute.

## Examples

```
time = as.POSIXct("2020-01-01 05:30:00", tz = "UTC")
acti_time_of_day(time, timezone = "America/New_York")
```

---

# acti_time_to_minute *Convert a Time of Day to a Minute Index*

---

## Description

Convert a Time of Day to a Minute Index

## Usage

```
acti_time_to_minute(x, start = 1L, timezone = NULL)
```

## Arguments

| | |
|---|---|
| x | An ‘hms‘, ‘POSIXt‘, or ‘HH:MM[:SS]‘ character vector. |
| start | The index assigned to midnight, either ‘0‘ or ‘1‘. |
| timezone | Optional Olson timezone for ‘POSIXt‘ values. |

## Value

An integer minute-of-day index.

## Note

This converts each minute of a day to either ‘0‘ through ‘1439‘ or ‘1‘ through ‘1440‘, the two common minute-of-day conventions.

## Examples

```
acti_time_to_minute(c("00:00", "12:00", "23:59"))
acti_time_to_minute("00:00", start = 0)
```

## as_convert_safe *Convert vectors ensuring no new NA*

### Description

Convert vectors ensuring no new NA

### Usage

```
as_convert_safe(x, ..., func = lubridate::as_datetime)

as_date_safe(x, ...)

as_datetime_safe(x, ...)
```

### Arguments

| | |
|---|---|
| x | a vector |
| ... | additional arguments to pass to ‘func‘ |
| func | the function to use to transform the vector ‘x‘ |

### Value

A converted ‘vector‘ the same length as ‘x‘ or errors if there are introduced NAs.

## create_day_inclusion *Create Day-Level Inclusion information*

### Description

Create Day-Level Inclusion information

### Usage

```
create_day_inclusion(data, min_required = 1368L)

add_day_inclusion(data, ...)
```

### Arguments

| | |
|---|---|
| data | A ‘data.frame‘ with the columns ‘time‘ |
| min_required | Number of minutes required in a day to be called ‘included‘ |
| ... | arguments to pass to [create_day_inclusion] when using [add_day_inclusion] |

## Value

A 'data.frame' for each day with information of number of minutes observed and included

## Examples

```
data = acti_raw_data |>
  dplyr::mutate(r = sqrt(X^2 + Y^2 + Z^2),
                time = lubridate::floor_date(time, "1 minute")) |>
  dplyr::group_by(time) |>
  dplyr::summarise(r = sum(r), .groups = "drop") |>
  dplyr::mutate(wear = r > 4000)

res = create_day_inclusion(data)
```

---

# flag_qc *Flag Quality Control Values*

---

## Description

Flag Quality Control Values

## Usage

```
flag_qc(
  df,
  dynamic_range = NULL,
  verbose = TRUE,
  flags = c("all", "spike", "interval_jump", "spike_second", "same_value",
    "device_limit", "all_zero", "impossible")
)

flag_qc_all(
  df,
  dynamic_range = NULL,
  verbose = TRUE,
  flags = c("all", "spike", "interval_jump", "spike_second", "same_value",
    "device_limit", "all_zero", "impossible")
)
```

## Arguments

| | |
|---|---|
| df | A data set of actigraphy |
| dynamic_range | dynamic range of the device, used to find the device limit. |
| verbose | print diagnostic messages |
| flags | the flags to run for QC. If you set this to "all", then all flags are run, as the default. |

## Value

A data set with a ‘flags‘ column (‘flag_qc‘) or a number of columns starting with ‘flag_*‘ (‘flag_qc_all‘)

## Examples

```
res = acti_raw_data
out = flag_qc(res)
```

---

`flag_spike` *Flag Spikes*

---

## Description

Flag Spikes

## Usage

```
flag_spike(df, spike_size = 11)

flag_interval_jump(df, verbose = FALSE)

flag_spike_second(df, spike_size = 11)

flag_device_limit(df, dynamic_range = NULL, epsilon = 0.05)

flag_contiguous_device_limit(df, dynamic_range = NULL, epsilon = 0.05)

flag_same_value(df, min_length = 1)

flag_all_zero(df, min_length = 3)

flag_impossible(df, min_length = 6)
```

## Arguments

| | |
|---|---|
| `df` | A data set of actigraphy |
| `spike_size` | size of "spike" - which is the absolute difference in contiguous observations on a single axis |
| `verbose` | print diagnostic messages |
| `dynamic_range` | dynamic range of the device, used to find the device limit. |
| `epsilon` | A small adjustment so that if values are within the device limit, but minus epsilon, still flagged as hitting the limit. For example, if ‘dynamic_range = c(-6, 6)‘ and ‘epsilon = 0.05‘, then any value <= ‘-5.95‘ or ‘>= 5.95‘ gravity units will be flagged |
| `min_length` | minimum length of the condition for contiguous samples. If ‘min_length = 3‘, then at least 3 ‘TRUE‘s in a row is required, any stretches of single ‘TRUE‘ values or 2 ‘TRUE‘ followed by ‘FALSE‘, will be set to ‘FALSE‘. |

## Value

A data set back

## Note

'flag_spike' looks if 2 contiguous values, within each axis, are larger than a absolute size ('11' gravity units). The 'flag_spike_second' function groups the data by second, finds the range of values, within each axis, and determines if this range is greater than a specified size ('11' g).

## Source

https://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/PAXMIN_G.htm

## Examples

```
res = acti_raw_data
res = flag_spike(res)
res = flag_interval_jump(res)
res = flag_spike_second(res)
res = flag_same_value(res)
res = flag_device_limit(res)
res = flag_all_zero(res)
res = flag_impossible(res)
```

---

# get_dynamic_range *Get Dynamic Range*

---

## Description

Get Dynamic Range

## Usage

```
get_dynamic_range(data, dynamic_range = NULL, flag_estimated = FALSE)
```

## Arguments

| | |
|---|---|
| data | An AccData object from an actigraphy reader |
| dynamic_range | the dynamic range. If this is not NULL, then it will be guess from the header or the data |
| flag_estimated | if 'TRUE', then the output will have the attribute '"estimated"', which is a logical indicated if it was found or estimated |

## Value

A length-2 numeric vector, or the original dynamic range (no checking done)

## get_sample_rate *Get Sample Rate*

### Description

Get Sample Rate

### Usage

```
get_sample_rate(data, sample_rate = NULL, flag_estimated = FALSE)
```

### Arguments

| | |
|---|---|
| data | A data set of actigraphy/activity data |
| sample_rate | the sample rate. If this is not NULL, then it will be guess from the header or the data or the data separation |
| flag_estimated | if 'TRUE', then the output will have the attribute '"estimated"', which is a logical indicated if it was found or estimated |

### Value

A length-1 numeric vector

## get_transformations *Get Transformations*

### Description

Get Transformations

### Usage

```
get_transformations(data)

prefix_transformations(transformations, prefix = NULL)

set_transformations(data, transformations, add = TRUE, prefix = NULL)
```

### Arguments

| | |
|---|---|
| data | data set of data, usually time and X/Y/Z. |
| transformations | character string of transformations |
| prefix | if not 'NULL', the prefix plus ':' would be pasted to the transformations. |
| add | Add the transformations to those already there in 'data' |

### Value

`set_transformations` returns the data, with the ‘transformations‘ attribute updated and `set_transformations` returns the attribute ‘transformations‘

---

## is.AccData *Is the object of class ‘AccData‘*

---

### Description

Is the object of class ‘AccData‘

### Usage

```
is.AccData(x)
```

### Arguments

| | |
|---|---|
| x | object to test |

### Value

Logical(1)

### Examples

```
is.AccData(mtcars)
```

---

## mark_condition *Mark a Condition of a Specified Minimum Length*

---

### Description

Mark a Condition of a Specified Minimum Length

### Usage

```
mark_condition(x, min_length = 1)
```

### Arguments

| | |
|---|---|
| x | A logical vector |
| min_length | minimum length, contiguous `TRUE` values required |

### Value

A logical vector

## Examples

```
x = c(FALSE, TRUE, TRUE, FALSE, FALSE, rep(TRUE, 10), FALSE, rep(TRUE, 20))
mark_condition(x)
mark_condition(x, 2)
mark_condition(x, 5)
mark_condition(x, 15)
```

---

# strip_hour_shift *Strip Hour Shift from Character Time Vector*

---

## Description

Strip Hour Shift from Character Time Vector

## Usage

```
strip_hour_shift(x, max_index = 2L)
```

## Arguments

| | |
|---|---|
| x | character vector with times that may include hour shifts (e.g., "2019-01-01 12:00+03:00") |
| max_index | maximum index to grab the shift from after splitting on spaces, default is 2 (e.g., "2019-01-01 12:00") |

## Value

A character vector with the '+'/'-' hour shift removed

## Examples

```
strip_hour_shift(c("2019-01-01 12:00+03:00", "2019-01-01 12:00-04:00"))
```

---

# xyz *Vector of X, Y, Z, and maybe time*

---

## Description

Vector of X, Y, Z, and maybe time

## Usage

```
xyz

xyzt

txyz
```

**Format**

A vector with 3 or 4 elements, which are:

**X** value to extract X column

**Y** value to extract Y column

**Z** value to extract Y column

**time** value to extract time column

An object of class `character` of length 4.

An object of class `character` of length 4.

# Index