

Package: PhysioDevices (via r-universe)

October 3, 2026

Title Wearable and Lab-System Device Ingestion for PhysioExperiment Objects

Version 0.9.6

Description Readers that ingest consumer-wearable and wired laboratory device exports into PhysioExperiment / MultiRatePhysioExperiment objects. Wearables: Empatica E4 CSV sessions and EmbracePlus Avro containers (multi-rate ACC / BVP / EDA / TEMP / HR / IBI streams), Shimmer GSR3+ Consensys exports (GSR, PPG, accelerometer), and Polar heart-rate / R-R interval exports. Lab systems: BIOPAC AcqKnowledge '.acq' files (per-channel rates and markers), Delsys Trigno EMG + IMU exports (multi-rate streams with per-sensor metadata), and Xsens MVNX motion-capture XML (segment kinematics, joint angles, centre of mass). Each import records a provenance entry and the correct per-signal sampling rates.

Depends R (>= 4.2), PhysioExperiment (>= 2.0.0)

Imports S4Vectors, stats, utils

Suggests PhysioCore, SummarizedExperiment, testthat (>= 3.2.0), reticulate, xml2, jsonlite, read.gt3x, GGIRread, FITfileR, knitr, rmarkdown

Additional_repositories <https://grimbough.r-universe.dev>

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'PhysioDevices-package.R' 'io-empatica.R' 'io-shimmer.R' 'io-polar.R' 'io-biopac.R' 'io-delsys.R' 'io-xsens.R' 'io-applehealth.R' 'io-apple-ecg.R' 'io-fitbit.R' 'io-googlefit.R' 'io-healthconnect.R' 'io-actigraph.R' 'io-actiwatch.R' 'io-tcx.R' 'io-oura.R' 'io-fit.R' 'io-whoop.R' 'io-withings.R' 'io-samsung.R'

RoxygenNote 7.3.3

URL <https://github.com/x-biosignal/PhysioDevices>,
<https://x-biosignal.r-universe.dev/PhysioDevices>,
<https://x-biosignal.github.io/PhysioDevices/>

BugReports <https://github.com/x-biosignal/PhysioDevices/issues>

Config/pak/sysreqs zlib1g-dev

Repository <https://x-biosignal.r-universe.dev>

Date/Publication 2026-10-03 13:11:46 UTC

RemoteUrl <https://github.com/x-biosignal/PhysioDevices>

RemoteRef HEAD

RemoteSha 3771065afbd990e110a45d0a7ae8835e882bfed3

Contents

appleHealthExperiment	3
appleHealthSeries	3
appleHealthTypes	4
fitbitSeries	5
hasAvroBackend	6
hasBioread	6
readActiwatch	7
readAppleECG	8
readAppleHealth	9
readBIOPAC	10
readCWA	11
readDelsysTrigno	12
readEmbracePlusAvro	13
readEmpaticaE4	14
readFIT	15
readFitbit	16
readGENEActiv	17
readGoogleFit	18
readGPX	19
readGT3X	20
readHealthConnect	21
readOura	22
readPolar	23
readSamsungHealth	24
readShimmer	25
readTCX	26
readWhoop	27
readWithings	28
readXsensMVNX	29

Index	30
--------------	-----------

appleHealthExperiment *Build a PhysioExperiment from one numeric Apple Health modality*

Description

Apple Health samples are irregularly spaced, so the returned experiment is event-like: a single channel holding the values, samplingRate NA, and the per-sample timestamps stored in `metadata(pe)$times`.

Usage

```
appleHealthExperiment(x, type)
```

Arguments

`x` An apple_health object from `readAppleHealth()`.
`type` A numeric record type (e.g. "HeartRate", "OxygenSaturation").

Value

A PhysioExperiment with one channel of the modality's values.

See Also

[appleHealthSeries\(\)](#)

Examples

```
rec <- function(v, t) sprintf(paste0(
  '<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Watch" ',
  'unit="count/min" startDate="%s" endDate="%s" value="%d"/>'), t, t, v)
p <- tempfile(fileext = ".xml")
writeLines(c("<HealthData>", rec(61, "2023-05-01 07:00:00 +0000"),
            rec(64, "2023-05-01 07:01:00 +0000"), "</HealthData>"), p)
pe <- appleHealthExperiment(readAppleHealth(p), "HeartRate")
dim(pe)
```

appleHealthSeries *Extract one modality's samples from an Apple Health export*

Description

Extract one modality's samples from an Apple Health export

Usage

```
appleHealthSeries(x, type)
```

Arguments

x An apple_health object from `readAppleHealth()`.

type A (prefix-stripped) record type, e.g. "HeartRate", "HeartRateVariabilitySDNN", "OxygenSaturation", "SleepAnalysis".

Value

A data frame of that type's records (start, end, value, value_num, unit, source), time-ordered.

See Also

`appleHealthTypes()`, `appleHealthExperiment()`

Examples

```
rec <- function(v, t) sprintf(paste0(
  '<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Watch" ',
  'unit="count/min" startDate="%s" endDate="%s" value="%d"/>', t, t, v)
p <- tempfile(fileext = ".xml")
writeLines(c("<HealthData>", rec(61, "2023-05-01 07:00:00 +0000"),
            rec(64, "2023-05-01 07:01:00 +0000"), "</HealthData>"), p)
appleHealthSeries(readAppleHealth(p), "HeartRate")
```

appleHealthTypes	<i>Summarise the record types in an Apple Health export</i>
------------------	---

Description

Summarise the record types in an Apple Health export

Usage

```
appleHealthTypes(x)
```

Arguments

x An apple_health object from `readAppleHealth()`.

Value

A data frame with one row per type: n, unit, and the time from/to.

See Also

`readAppleHealth()`, `appleHealthSeries()`

Examples

```
rec <- function(v, t) sprintf(paste0(
  '<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Watch" ',
  'unit="count/min" startDate="%s" endDate="%s" value="%d"/>'), t, t, v)
p <- tempfile(fileext = ".xml")
writeLines(c("<HealthData>", rec(61, "2023-05-01 07:00:00 +0000"),
  rec(64, "2023-05-01 07:01:00 +0000"), "</HealthData>"), p)
appleHealthTypes(readAppleHealth(p))
```

fitbitSeries

Extract one modality from a Fitbit archive

Description

Extract one modality from a Fitbit archive

Usage

```
fitbitSeries(x, modality)
```

Arguments

x A fitbit object from `readFitbit()`.

modality One of the modality names present in x (e.g. "heart_rate", "spo2", "sleep").

Value

The modality's data frame.

See Also

`readFitbit()`

Examples

```
# readFitbit() guards on the jsonlite package (a Suggests dependency).
if (requireNamespace("jsonlite", quietly = TRUE)) {
  d <- tempfile("fitbit")
  dir.create(d)
  writelines(paste0('["dateTime":"05/01/23 07:00:00","value":{"bpm":61}},{',
    '{"dateTime":"05/01/23 07:01:00","value":{"bpm":64}}]'),
    file.path(d, "heart_rate.json"))
  fb <- readFitbit(d, what = "heart_rate")
  fitbitSeries(fb, "heart_rate")
}
```

hasAvroBackend	<i>Is an Avro backend available?</i>
----------------	--------------------------------------

Description

Is an Avro backend available?

Usage

```
hasAvroBackend()
```

Value

TRUE when the Python fastavro module is reachable via **reticulate**.

See Also

[readEmbracePlusAvro\(\)](#)

Examples

```
hasAvroBackend()
```

hasBioread	<i>Is a BIOPAC (bioread) backend available?</i>
------------	---

Description

Is a BIOPAC (bioread) backend available?

Usage

```
hasBioread()
```

Value

TRUE when the Python bioread module is reachable via **reticulate**.

See Also

[readBIOPAC\(\)](#)

Examples

```
hasBioread()
```

readActiwatch	<i>Read an Actiwatch AWD actigraphy file</i>
---------------	--

Description

Parses a Philips/Respironics Actiwatch .awd file into per-epoch activity counts with timestamps. The start date/time is read from the header; the activity counts are the longest run of numeric lines (the data block).

Usage

```
readActiwatch(path, epoch = 60, tz = "UTC")
```

Arguments

path	Path to a .awd file.
epoch	Epoch length in seconds (default 60). The .awd header does not reliably carry it, so set it to your recording's epoch.
tz	Time zone for the timestamps (default "UTC").

Value

A data frame with time (POSIXct) and activity (counts), with the epoch length in attr(, "epoch_sec"). Feeds `PhysioWearable::coleKripke()` (which expects 1-minute epochs).

References

Philips Respironics Actiwatch / Actiware AWD format.

See Also

[readGT3X\(\)](#), [readGENEActiv\(\)](#)

Examples

```
# An .awd is a short text header (date on line 2, time on line 3) then one
# activity count per line; write a tiny one and read it back (base-R parser).
p <- tempfile(fileext = ".awd")
writeLines(c("SUBJECT01", "01-May-2023", "23:00", "V", "M", "A", "W",
            as.character(c(12, 0, 5, 40, 3, 0, 0, 8, 22, 15, 7, 2, 0, 0, 33))), p)
aw <- readActiwatch(p, epoch = 60)
head(aw)
```

readAppleECG

Read an Apple Watch ECG recording

Description

Parses the single-lead ECG CSV that the Apple Watch ECG app exports into a `PhysioExperiment` (one ECG channel). The classification, recorded date and sample rate from the file header are kept in `metadata()`. The result feeds the `PhysioECG` beat-detection and heart-rate-variability functions.

Usage

```
readAppleECG(path, default_rate = 512)
```

Arguments

<code>path</code>	Path to an Apple Watch ECG .csv export.
<code>default_rate</code>	Sample rate (Hz) to assume if the header has none (default 512, the Apple Watch ECG rate).

Value

A `PhysioExperiment` with a raw assay of the ECG voltage (microvolts) and `samplingRate` set from the header.

References

Apple Inc. Taking an ECG with the ECG app on Apple Watch.

See Also

[readAppleHealth\(\)](#)

Examples

```
# The ECG app writes a "key,value" header then one voltage sample per line;
# write a tiny export and read it back (base-R parser, no toolchain needed).
p <- tempfile(fileext = ".csv")
writeLines(c("Name,Jane Doe", "Sample Rate,512 Hz", "Unit,uV",
             "Classification,Sinus Rhythm",
             format(round(sin(seq(0, 20 * pi, length.out = 1024)) * 300, 2))), p)
ecg <- readAppleECG(p)
samplingRate(ecg)
```

readAppleHealth	<i>Read an Apple Health export</i>
-----------------	------------------------------------

Description

Parses an Apple Health export.xml into a tidy set of records and workout summaries. Every sample Apple stores as a <Record> – heart rate, heart-rate variability (HeartRateVariabilitySDNN), blood oxygen (OxygenSaturation), respiratory rate, step count, active/basal energy, resting and walking heart rate, VO2 max, and SleepAnalysis sleep stages – becomes one row; <Workout> elements become the workouts table. The reader streams the file, so multi- hundred-MB exports are handled without loading the whole document into memory.

Usage

```
readAppleHealth(path, types = NULL, tz = "UTC")
```

Arguments

path	Path to the Apple Health export.xml.
types	Optional character vector of (prefix-stripped) record types to keep; NULL (default) keeps all.
tz	Time zone for the parsed timestamps (default "UTC").

Details

The Apple type prefix (HKQuantityTypeIdentifier / HKCategoryTypeIdentifier) is stripped, so type reads e.g. "HeartRate", "OxygenSaturation", "SleepAnalysis". Numeric samples are parsed into value_num; category samples (sleep stages) keep their (prefix-stripped) label in value.

Value

An apple_health object: a list with records (a data frame with type, source, unit, start, end, value, value_num), workouts (or NULL), path and tz.

References

Apple Inc. HealthKit data types and the Health app XML export.

See Also

[appleHealthSeries\(\)](#), [appleHealthTypes\(\)](#), [appleHealthExperiment\(\)](#), [readAppleECG\(\)](#)

Examples

```
# Apple exports each sample as a self-closing <Record> line; write a tiny
# export.xml and read it back (the parser is base-R, so no toolchain is needed).
p <- tempfile(fileext = ".xml")
writeLines(c(
  '<?xml version="1.0" encoding="UTF-8"?>', '<HealthData locale="en_US">',
  paste0('<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Watch"',
    ' unit="count/min" startDate="2023-05-01 07:00:00 +0000"',
    ' endDate="2023-05-01 07:00:00 +0000" value="61"/>'),
  paste0('<Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Watch"',
    ' unit="count/min" startDate="2023-05-01 07:01:00 +0000"',
    ' endDate="2023-05-01 07:01:00 +0000" value="64"/>'),
  '</HealthData>'), p)
ah <- readAppleHealth(p)
appleHealthTypes(ah)
appleHealthSeries(ah, "HeartRate")
```

readBIOPAC

Read a BIOPAC AcqKnowledge (.acq) file

Description

Parses a BIOPAC AcqKnowledge .acq file (via the Python bioread module through **reticulate**; see [hasBioread\(\)](#)) into a [PhysioExperiment::MultiPhysioExperiment](#). Each acquisition channel becomes a stream at its own sample rate (channels may be acquired at different rates via frequency dividers), carrying its label and unit. Event markers become [PhysioExperiment::PhysioEvents](#) on the fastest stream.

Usage

```
readBIOPAC(path)
```

Arguments

path Path to a .acq file.

Value

A [PhysioExperiment::MultiPhysioExperiment](#).

References

bioread: <https://github.com/uwmadison-chm/bioread>.

See Also

[hasBioread\(\)](#), [readDelsysTrigno\(\)](#), [readXsensMVNX\(\)](#)

Examples

```
# .acq is a binary format: the package ships a small sample and the parse needs
# the Python 'bioread' backend, so both are checked before reading.
f <- system.file("extdata", "biopac_sample.acq", package = "PhysioDevices")
if (nzchar(f) && hasBioread()) {
  mr <- readBIOPAC(f)
  PhysioExperiment::streamNames(mr)
}
```

readCWA

*Read an Axivity AX3/AX6 CWA accelerometer file***Description**

Reads a raw Axivity .cwa file (the accelerometer format used by UK Biobank and other large cohorts) into a tri-axial-acceleration PhysioExperiment via the **GGIRread** package, in the same shape as [readGT3X\(\)](#) / [PhysioWearable::accelToPhysioExperiment\(\)](#).

Usage

```
readCWA(path, start = 1, end = NULL)
```

Arguments

path	Path to a .cwa file.
start, end	Block range to read (1-based). end = NULL (default) reads the whole file. For very long recordings, read in ranges.

Value

A PhysioExperiment with an acceleration assay (x/y/z, g) at the device sample rate.

References

Axivity AX3/AX6; GGIRread::readAxivity.

See Also

[readGT3X\(\)](#), [PhysioWearable::readAccelCSV\(\)](#)

Examples

```
## Not run:
# .cwa is a bit-packed Axivity binary with no R writer and no bundled sample,
# so this needs a real device file plus the GGIRread backend; not runnable
# offline.
pe <- readCWA("subject.cwa")
acc <- SummarizedExperiment::assay(pe, "acceleration")
```

```

enmo <- PhysioWearable::computeENMO(acc, sampling_rate = samplingRate(pe))

## End(Not run)

```

readDelsysTrigno	<i>Read a Delsys Trigno CSV export</i>
------------------	--

Description

Parses a Delsys Trigno File Utility / EMGworks CSV export into a [PhysioExperiment::MultiPhysioExperiment](#). EMG channels (high rate) and the IMU channels (accelerometer / gyroscope / magnetometer, lower rate) are separated into their own streams, each at the rate implied by its per-channel X[s] time column, with per-sensor column metadata (sensor number and modality).

Usage

```
readDelsysTrigno(path, sep = ",")
```

Arguments

path	Path to a Trigno .csv export.
sep	Field separator (default: comma).

Value

A [PhysioExperiment::MultiPhysioExperiment](#) with an "emg" stream and, when present, an "imu" stream.

References

Delsys. "Trigno File Utility / EMGworks export."

See Also

[readXsensMVNX\(\)](#), [readBIOPAC\(\)](#)

Examples

```

f <- system.file("extdata", "delsys_trigno.csv", package = "PhysioDevices")
if (nzchar(f)) {
  mr <- readDelsysTrigno(f)
  PhysioExperiment::streamNames(mr)
}

```

readEmbracePlusAvro	<i>Read an Empatica EmbracePlus Avro container</i>
---------------------	--

Description

Decodes an EmbracePlus .avro file into a [PhysioExperiment::MultiPhysioExperiment](#). Each signal in the record's rawData (EDA, BVP, temperature, accelerometer) becomes a stream at its samplingFrequency, aligned by its timestampStart (microseconds). Requires an Avro backend (the Python fastavro module via **reticulate**; see [hasAvroBackend\(\)](#)).

Usage

```
readEmbracePlusAvro(path)
```

Arguments

path	Path to an EmbracePlus .avro file.
------	------------------------------------

Value

A [PhysioExperiment::MultiPhysioExperiment](#).

References

Empatica. "EmbracePlus — Avro data format."

See Also

[readEmpaticaE4\(\)](#), [hasAvroBackend\(\)](#)

Examples

```
# .avro is a binary container: the package ships a small sample and decoding
# needs the Python 'fastavro' backend, so both are checked before reading.
f <- system.file("extdata", "embraceplus_sample.avro", package = "PhysioDevices")
if (nzchar(f) && hasAvroBackend()) {
  mr <- readEmbracePlusAvro(f)
  PhysioExperiment::streamNames(mr)
}
```

readEmpaticaE4	<i>Read an Empatica E4 CSV session directory</i>
----------------	--

Description

Reads an Empatica E4 session (a directory of per-signal CSV files) into a [PhysioExperiment::MultiPhysioExperiment](#). Each present signal (EDA at 4 Hz, BVP at 64 Hz, ACC at 32 Hz, TEMP at 4 Hz, HR at 1 Hz) becomes its own stream at its native rate, and the per-signal UTC start times are aligned on a common clock via per-stream offsets. Inter-beat intervals (`IBI.csv`) and event tags (`tags.csv`) are attached as events.

Usage

```
readEmpaticaE4(dir)
```

Arguments

`dir` Path to an E4 session directory.

Value

A [PhysioExperiment::MultiPhysioExperiment](#) with one stream per signal.

References

Empatica. "E4 wristband data — exported CSV file format."

See Also

[readEmbracePlusAvro\(\)](#), [readShimmer\(\)](#)

Examples

```
d <- system.file("extdata", "e4_sample", package = "PhysioDevices")
if (nzchar(d)) {
  mr <- readEmpaticaE4(d)
  PhysioExperiment::streamNames(mr)
}
```

readFIT	<i>Read a Garmin/ANT+ FIT workout file</i>
---------	--

Description

Reads a .fit file (Garmin, Wahoo, Coros, ...) into a tidy trackpoint data frame, in the same shape as [readTCX\(\)](#) / [readGPX\(\)](#). The binary decode is delegated to the FITfileR package (not on CRAN – install with `install.packages("FITfileR", repos = "https://grimbough.r-universe.dev")`).

Usage

```
readFIT(path, tz = "UTC")
```

Arguments

path	Path to a .fit file.
tz	Time zone for the parsed timestamps (default "UTC").

Value

A data frame of trackpoints (time, lat, lon, altitude_m, distance_m, hr, cadence, speed, power), time-ordered.

References

Garmin FIT SDK; grimbough/FITfileR.

See Also

[readTCX\(\)](#), [readGPX\(\)](#)

Examples

```
## Not run:
# .fit is a binary Garmin/ANT+ format with no R writer and no bundled sample,
# so this needs a real device file plus the FITfileR backend (install from
# the grimbough r-universe); not runnable offline.
tp <- readFIT("activity.fit")

## End(Not run)
```

readFitbit

*Read a Fitbit / Google Pixel data archive***Description**

Parses a Fitbit account-export (or Google Takeout) directory into tidy per-modality series. Files are discovered by name and both layouts are handled: the account archive's per-day files (heart_rate-YYYY-MM-DD.json, steps-*, sleep-*, ...) and the Web-API / research extraction's aggregated single files (heart_rate.json, steps.json, sleep.json; e.g. the PMData dataset). Modalities: heart rate (bpm), steps, resting_heart_rate, SpO2 (spo2*/oxygen*, JSON or minute CSV), and sleep (Fitbit's own wake/light/deep/rem stages). Google Pixel Watch health data is stored in Fitbit, so the same reader covers it.

Usage

```
readFitbit(path, what = NULL, tz = "UTC")
```

Arguments

path	Directory of the extracted archive (searched recursively), or a file / vector of files.
what	Optional character vector restricting which modalities to read (any of "heart_rate", "steps", "spo2", "resting_heart_rate", "sleep"); NULL (default) reads all that are found.
tz	Time zone for the parsed timestamps (default "UTC").

Value

A fitbit object: a list with any of heart_rate, steps, spo2, resting_heart_rate (data frames of time, value), sleep (stage intervals start, end, stage, log_id) and sleep_summary (Fitbit's per-log efficiency, minutes_asleep, time_in_bed, ...), plus path/tz. The sleep stages feed `PhysioWearable::summarizeSleepStages()`; spo2 feeds `PhysioWearable::spo2Metrics()`.

References

Fitbit Web API data types; Fitbit account data export.

See Also

[readAppleHealth\(\)](#), [fitbitSeries\(\)](#)

Examples

```
# Reading Fitbit needs the jsonlite package (a Suggests dependency); write a
# tiny aggregated heart_rate.json into a temp dir and read it back.
if (requireNamespace("jsonlite", quietly = TRUE)) {
  d <- tempfile("fitbit")
  dir.create(d)
  writeLines(paste0('{"dateTime":"05/01/23 07:00:00","value":{"bpm":61}}',
    ,
```

```

      '{"dateTime":"05/01/23 07:01:00","value":{"bpm":64}}']'),
      file.path(d, "heart_rate.json"))
  fb <- readFitbit(d, what = "heart_rate")
  fb
}

```

readGENEActiv

*Read a GENEActiv .bin accelerometer file***Description**

Reads a raw GENEActiv .bin file (another research accelerometer used in large cohorts) into a tri-axial-acceleration PhysioExperiment, in the same shape as [readGT3X\(\)](#) / [readCWA\(\)](#), via the **GGIRread** package.

Usage

```
readGENEActiv(path, start = 1, end = NULL, sampling_rate = NULL)
```

Arguments

path	Path to a .bin file.
start, end	Page range to read (1-based). end = NULL (default) reads the whole file.
sampling_rate	Optional sample-rate override (Hz), used when the header does not report it; otherwise taken from the header or the timestamps.

Value

A PhysioExperiment with an acceleration assay (x/y/z, g).

References

GENEActiv; GGIRread::readGENEActiv.

See Also

[readCWA\(\)](#), [readGT3X\(\)](#)

Examples

```

## Not run:
# .bin is a bit-packed GENEActiv binary with no R writer and no bundled sample,
# so this needs a real device file plus the GGIRread backend; not runnable
# offline.
pe <- readGENEActiv("subject.bin")

## End(Not run)

```

readGoogleFit

Read a Google Fit (Google Takeout) export

Description

Parses the "Fit" folder of a Google Takeout archive. From "All Data" it reads the granular per-type JSON points – heart rate (`heart_rate.bpm`) and steps (`step_count.delta`) – decoding their nanosecond timestamps; from "Daily activity metrics" it reads the daily-summary CSVs (steps, calories, distance, average/max/min heart rate).

Usage

```
readGoogleFit(path, what = NULL, tz = "UTC")
```

Arguments

<code>path</code>	Directory of the extracted Takeout "Fit" folder (searched recursively), or a file / vector of files.
<code>what</code>	Optional restriction, any of "heart_rate", "steps", "daily".
<code>tz</code>	Time zone for parsed timestamps (default "UTC").

Value

A `google_fit` object: a list with any of `heart_rate`, `steps` (time, value) and `daily` (per-day summary), plus `path/tz`.

References

Google Takeout "Fit" export; Google Fit data types.

See Also

[readHealthConnect\(\)](#), [readFitbit\(\)](#)

Examples

```
# readGoogleFit() guards on the jsonlite package (a Suggests dependency). Its
# "Daily activity metrics" CSV path is a self-contained round trip: write one
# daily-summary CSV under that folder name and read it back.
if (requireNamespace("jsonlite", quietly = TRUE)) {
  d <- tempfile("Fit")
  dir.create(file.path(d, "Daily activity metrics"), recursive = TRUE)
  write.csv(data.frame("Date" = "2023-05-01", "Step count" = 8210,
    "Calories (kcal)" = 2110, "Distance (m)" = 6400,
    check.names = FALSE),
    file.path(d, "Daily activity metrics", "2023-05-01.csv"),
    row.names = FALSE)
  gf <- readGoogleFit(d, what = "daily")
}
```

```
    gf$daily
  }
```

readGPX

*Read a GPX (GPS Exchange) track file***Description**

Parses the track points of a .gpx file (a widely shared GPS/workout export) into a tidy data frame: time, position, altitude, and heart rate / cadence when the Garmin TrackPointExtension is present.

Usage

```
readGPX(path, tz = "UTC")
```

Arguments

path Path to a .gpx file.
tz Time zone for the parsed timestamps (default "UTC").

Value

A data frame of track points (time, lat, lon, altitude_m, hr, cadence), time-ordered.

References

GPX 1.1 schema; Garmin TrackPointExtension.

See Also

[readTCX\(\)](#)

Examples

```
# Reading GPX needs the xml2 package (a Suggests dependency); write a tiny
# two-point .gpx and read it back.
if (requireNamespace("xml2", quietly = TRUE)) {
  p <- tempfile(fileext = ".gpx")
  writelines(c(
    '<gpx version="1.1"><trk><trkseg>',
    paste0('<trkpt lat="35.17" lon="136.96"><ele>12</ele>',
    '<time>2023-05-01T07:00:00Z</time></trkpt>'),
    paste0('<trkpt lat="35.1701" lon="136.9601"><ele>13</ele>',
    '<time>2023-05-01T07:00:05Z</time></trkpt>'),
    '</trkseg></trk></gpx>'), p)
  readGPX(p)
}
```

readGT3X

*Read an ActiGraph GT3X accelerometer file***Description**

Reads a raw ActiGraph .gt3x file (the common US physical-activity / actigraphy device format) into a tri-axial-acceleration `PhysioExperiment`. The binary decode is done by the **read.gt3x** package; the result is the same shape as `PhysioWearable::accelToPhysioExperiment()`, so it feeds the free-living accelerometry pipeline (`computeENMO()`, `summarizeFreeLiving()`).

Usage

```
readGT3X(path, imputeZeroes = TRUE)
```

Arguments

<code>path</code>	Path to a .gt3x file.
<code>imputeZeroes</code>	If TRUE (default) idle-sleep-mode gaps are filled with zeroes (passed to <code>read.gt3x::read.gt3x()</code>) so the series stays regularly sampled.

Value

A `PhysioExperiment` with an acceleration assay (x/y/z, g) at the device sample rate; device metadata in `metadata()`.

References

ActiGraph GT3X format; Neishabouri A et al. `read.gt3x`.

See Also

`readCWA()`, `PhysioWearable::readAccelCSV()`

Examples

```
## Not run:
# .gt3x is a bit-packed binary format with no R writer and no bundled sample,
# so this needs a real ActiGraph device file plus the read.gt3x backend and
# cannot be run offline.
pe <- readGT3X("subject.gt3x")
acc <- SummarizedExperiment::assay(pe, "acceleration")
PhysioWearable::computeENMO(acc, sampling_rate = samplingRate(pe))

## End(Not run)
```

readHealthConnect	<i>Read an Android Health Connect CSV export</i>
-------------------	--

Description

Parses a directory of Health Connect record-type CSVs (one file per type) into tidy per-modality series. Recognises HeartRate, Steps, OxygenSaturation, RespiratoryRate and SleepSession (its stages) by filename; the time and value columns are found by name heuristics, so the common exporter layouts work without configuration.

Usage

```
readHealthConnect(path, tz = "UTC")
```

Arguments

path	Directory of the CSV export (searched recursively), or a file / vector of files.
tz	Time zone for parsed timestamps (default "UTC").

Value

A health_connect object: a list with any of heart_rate, steps, spo2, respiratory_rate (time, value), and sleep (stage intervals start, end, stage), plus path/tz. spo2 feeds `PhysioWearable::spo2Metrics()`, sleep feeds `PhysioWearable::summarizeSleepStages()`.

References

Android Health Connect data types and export.

See Also

[readGoogleFit\(\)](#), [readFitbit\(\)](#)

Examples

```
# Health Connect exports one CSV per record type (type taken from the
# filename); write a tiny HeartRate.csv into a temp dir and read it back.
d <- tempfile("healthconnect")
dir.create(d)
write.csv(data.frame(Time = c("2023-05-01T07:00:00Z", "2023-05-01T07:01:00Z"),
                      BeatsPerMinute = c(61, 64), check.names = FALSE),
          file.path(d, "HeartRate.csv"), row.names = FALSE)
hc <- readHealthConnect(d)
hc$heart_rate
```

readOura	<i>Read an Oura Ring export</i>
----------	---------------------------------

Description

Parses an Oura sleep export/JSON (API v2 sleep collection, or the older API v1 sleep) into tidy per-night data: sleep-stage intervals (from the 5-minute phase string), the 5-minute heart-rate and HRV (RMSSD) series, and Oura's own sleep summary. Both API versions are recognised.

Usage

```
readOura(path, tz = "UTC")
```

Arguments

path	Path to the Oura sleep JSON (a v2 object with a data array, a v1 object with a sleep array, or a bare array of sleep periods).
tz	Time zone for the parsed timestamps (default "UTC").

Value

An oura object: a list with sleep (stage intervals start, end, stage, day), sleep_summary (per-day efficiency, total_sleep_min, time_in_bed_min), heart_rate (time, bpm) and hrv (time, rmssd), plus path/tz. The sleep stages feed `PhysioWearable::summarizeSleepStages()`.

References

Oura API v2 sleep data.

See Also

[readFitbit\(\)](#), [readHealthConnect\(\)](#)

Examples

```
# Reading Oura needs the jsonlite package (a Suggests dependency); write a tiny
# API-v1 sleep JSON and read it back.
if (requireNamespace("jsonlite", quietly = TRUE)) {
  p <- tempfile(fileext = ".json")
  rec <- list(day = "2023-05-01", bedtime_start = "2023-05-01T23:00:00+00:00",
             hypnogram_5min = "1223334444", efficiency = 90,
             total_sleep_duration = 27000, time_in_bed = 30000,
             hr_5min = c(58, 57, 56, 55, 60, 62, 59, 58, 57, 61),
             rmssd_5min = c(40, 42, 45, 50, 48, 30, 35, 38, 41, 39))
  jsonlite::write_json(list(sleep = list(rec)), p, auto_unbox = TRUE)
  ou <- readOura(p)
  ou$sleep_summary
}
```

readPolar	<i>Read a Polar heart-rate / R-R interval export</i>
-----------	--

Description

Reads a Polar (Flow / H10) export into a `PhysioExperiment`. A heart-rate column (bpm) becomes the "raw" assay; R-R intervals (ms) are attached as [PhysioExperiment::PhysioEvents](#) (one event per beat, with onset at the cumulative R-R time and the interval in the value). When only R-R intervals are present, instantaneous heart rate ($60000 / \text{R-R}$) is used as the signal.

Usage

```
readPolar(path, sep = ",")
```

Arguments

path	Path to a Polar .csv/.txt export.
sep	Field separator (default: comma).

Value

A `PhysioExperiment`; R-R intervals are available via [PhysioExperiment::getEvents\(\)](#).

References

Polar. "Flow / H10 heart-rate and R-R export."

See Also

[readEmpaticaE4\(\)](#), [readShimmer\(\)](#)

Examples

```
f <- system.file("extdata", "polar_hr.csv", package = "PhysioDevices")
if (nzchar(f)) {
  pe <- readPolar(f)
  PhysioExperiment::nEvents(PhysioExperiment::getEvents(pe))
}
```

readSamsungHealth	<i>Read a Samsung Health export</i>
-------------------	-------------------------------------

Description

Parses a Samsung Health CSV export directory into tidy per-modality series – heart rate, blood oxygen (SpO2) and steps – detecting the record type from the `com.samsung...` filename and the time/value columns heuristically (each file has a one-line metadata comment above its header). Galaxy Watch health data is stored in Samsung Health, so this covers it.

Usage

```
readSamsungHealth(path, tz = "UTC")
```

Arguments

<code>path</code>	Directory of the export (searched recursively), or a file / vector of files.
<code>tz</code>	Time zone for the parsed timestamps (default "UTC").

Value

A `samsung_health` object: a list with any of `heart_rate` (bpm), `spo2`, `steps` (time, value), plus `path/tz`. `spo2` feeds `PhysioWearable::spo2Metrics()`.

References

Samsung Health data export.

See Also

[readHealthConnect\(\)](#), [readFitbit\(\)](#)

Examples

```
# Samsung Health CSVs carry a one-line metadata comment above the header (so
# they are read with skip = 1); write a tiny heart_rate file and read it back.
d <- tempfile("samsung")
dir.create(d)
writeLines(c("com.samsung.shealth.tracker.heart_rate,2,...",
             "start_time,heart_rate",
             "2023-05-01 07:00:00,61", "2023-05-01 07:01:00,64"),
           file.path(d, "com.samsung.shealth.tracker.heart_rate.202305.csv"))
sh <- readSamsungHealth(d)
sh$heart_rate
```

readShimmer	<i>Read a Shimmer GSR3+ Consensys CSV export</i>
-------------	--

Description

Reads a Shimmer GSR3+ Consensys export (GSR / skin conductance, PPG, accelerometer) into a `PhysioExperiment`. The calibrated (`_CAL`) columns are read in their physical units; the timestamp column determines the sampling rate; and each channel's unit (from the export's unit row) is stored in the column metadata.

Usage

```
readShimmer(path, sep = NULL)
```

Arguments

<code>path</code>	Path to a Shimmer Consensys .csv file.
<code>sep</code>	Field separator. By default it is taken from a leading <code>sep=<char></code> line, falling back to a tab.

Value

A `PhysioExperiment` with one channel per calibrated signal.

References

Shimmer. "Consensys — data export format."

See Also

[readEmpaticaE4\(\)](#), [readPolar\(\)](#)

Examples

```
f <- system.file("extdata", "shimmer_gsr3.csv", package = "PhysioDevices")
if (nzchar(f)) {
  pe <- readShimmer(f)
  channelNames(pe)
}
```

readTCX

*Read a TCX (Garmin Training Center) workout file***Description**

Parses the trackpoints of a .tcx workout (the common Garmin Connect / Strava export) into a tidy data frame: time, position, altitude, distance, heart rate and cadence/speed where present.

Usage

```
readTCX(path, tz = "UTC")
```

Arguments

path Path to a .tcx file.

tz Time zone for the parsed timestamps (default "UTC").

Value

A data frame of trackpoints (time, lat, lon, altitude_m, distance_m, hr, cadence, speed), time-ordered, with the activity sport in attr(, "sport").

References

Garmin Training Center XML (TCX) schema.

See Also

[readGPX\(\)](#), [readFitbit\(\)](#)

Examples

```
# Reading TCX needs the xml2 package (a Suggests dependency); write a tiny
# two-trackpoint .tcx and read it back.
if (requireNamespace("xml2", quietly = TRUE)) {
  p <- tempfile(fileext = ".tcx")
  writeLines(c(
    '<TrainingCenterDatabase><Activities><Activity Sport="Running"><Lap><Track>',
    paste0('<Trackpoint><Time>2023-05-01T07:00:00Z</Time><Position>',
      '<LatitudeDegrees>35.17</LatitudeDegrees>',
      '<LongitudeDegrees>136.96</LongitudeDegrees></Position>',
      '<DistanceMeters>0</DistanceMeters>',
      '<HeartRateBpm><Value>96</Value></HeartRateBpm></Trackpoint>'),
    paste0('<Trackpoint><Time>2023-05-01T07:00:05Z</Time><Position>',
      '<LatitudeDegrees>35.1701</LatitudeDegrees>',
      '<LongitudeDegrees>136.9601</LongitudeDegrees></Position>',
      '<DistanceMeters>9</DistanceMeters>',
      '<HeartRateBpm><Value>99</Value></HeartRateBpm></Trackpoint>'),
    '</Track></Lap></Activity></Activities></TrainingCenterDatabase>'), p)
```

```

    readTCX(p)
  }

```

readWhoop

Read a Whoop data export

Description

Parses a Whoop CSV export directory into tidy daily-summary tables: sleep (per-night stage durations, efficiency, HRV, respiratory rate) and recovery (recovery score, resting heart rate, HRV, blood oxygen, skin temperature). Whoop exports summaries, not epoch-level stages.

Usage

```
readWhoop(path)
```

Arguments

path Directory of the Whoop export (searched recursively), or a file / vector of files.

Value

A whoop object: a list with sleep and/or recovery data frames, plus path.

References

Whoop data export.

See Also

[readOura\(\)](#), [readFitbit\(\)](#)

Examples

```

# Whoop exports a folder of summary CSVs; write a tiny physiological_cycles.csv
# into a temp dir and read it back (base-R read.csv, no toolchain needed).
d <- tempfile("whoop")
dir.create(d)
write.csv(data.frame("Cycle start time" = "2023-05-01", "Recovery score %" = 66,
  "Resting heart rate (bpm)" = 54, "Heart rate variability (ms)" = 48,
  "Blood oxygen %" = 97, "Skin temp (celsius)" = 33.2, check.names = FALSE),
  file.path(d, "physiological_cycles.csv"), row.names = FALSE)
w <- readWhoop(d)
w$recovery

```

readWithings

Read a Withings Health Mate export

Description

Parses a Withings "Download my data" export directory into tidy per-modality tables: heart_rate (bpm), spo2, blood_pressure (systolic, diastolic) and weight (kg). The modality is taken from the filename and the time/value columns are detected heuristically. Covers Withings Scan-Watch / Sleep / BPM / Body devices.

Usage

```
readWithings(path, tz = "UTC")
```

Arguments

path	Directory of the export (searched recursively), or a file / vector of files.
tz	Time zone for the parsed timestamps (default "UTC").

Value

A withings object: a list with any of heart_rate, spo2, blood_pressure, weight, plus path/tz. spo2 feeds `PhysioWearable::spo2Metrics()`.

References

Withings Health Mate data export.

See Also

[readOura\(\)](#), [readFitbit\(\)](#)

Examples

```
# Withings exports per-modality CSVs (modality taken from the filename); write
# a tiny weight.csv into a temp dir and read it back (base-R read.csv).
d <- tempfile("withings")
dir.create(d)
write.csv(data.frame(date = c("2023-05-01", "2023-05-08"),
                     "Weight (kg)" = c(70.2, 69.8), check.names = FALSE),
          file.path(d, "weight.csv"), row.names = FALSE)
w <- readWithings(d)
w$weight
```

`readXsensMVNX`*Read an Xsens MVNX motion-capture export*

Description

Parses an Xsens MVN Analyze .mvnx XML file into a [PhysioExperiment::MultiPhysioExperiment](#). Each kinematic signal present in the "normal" frames (segment position, orientation, velocity, acceleration, angular velocity/acceleration, per-joint angles, and centre of mass) becomes a stream at the subject frame rate, with per-channel labels built from the segment/joint names.

Usage

```
readXsensMVNX(path)
```

Arguments

path	Path to a .mvnx file.
------	-----------------------

Value

A [PhysioExperiment::MultiPhysioExperiment](#).

References

Xsens. "MVNX file format (MVN Analyze)."

See Also

[readBIOPAC\(\)](#), [readDelsysTrigno\(\)](#)

Examples

```
f <- system.file("extdata", "xsens_sample.mvnx", package = "PhysioDevices")
if (nzchar(f) && requireNamespace("xml2", quietly = TRUE)) {
  mr <- readXsensMVNX(f)
  PhysioExperiment::streamNames(mr)
}
```

Index

appleHealthExperiment, 3
appleHealthExperiment(), 4, 9
appleHealthSeries, 3
appleHealthSeries(), 3, 4, 9
appleHealthTypes, 4
appleHealthTypes(), 4, 9

fitbitSeries, 5
fitbitSeries(), 16

hasAvroBackend, 6
hasAvroBackend(), 13
hasBioread, 6
hasBioread(), 10

PhysioExperiment::getEvents(), 23
PhysioExperiment::MultiPhysioExperiment,
10, 12–14, 29
PhysioExperiment::PhysioEvents, 10, 23
PhysioWearable::accelToPhysioExperiment(),
11, 20
PhysioWearable::readAccelCSV(), 11, 20

read.gt3x::read.gt3x(), 20
readActiwatch, 7
readAppleECG, 8
readAppleECG(), 9
readAppleHealth, 9
readAppleHealth(), 3, 4, 8, 16
readBIOPAC, 10
readBIOPAC(), 6, 12, 29
readCWA, 11
readCWA(), 17, 20
readDelsysTrigno, 12
readDelsysTrigno(), 10, 29
readEmbracePlusAvro, 13
readEmbracePlusAvro(), 6, 14
readEmpaticaE4, 14
readEmpaticaE4(), 13, 23, 25
readFIT, 15

readFitbit, 16
readFitbit(), 5, 18, 21, 22, 24, 26–28
readGENEActiv, 17
readGENEActiv(), 7
readGoogleFit, 18
readGoogleFit(), 21
readGPX, 19
readGPX(), 15, 26
readGT3X, 20
readGT3X(), 7, 11, 17
readHealthConnect, 21
readHealthConnect(), 18, 22, 24
readOura, 22
readOura(), 27, 28
readPolar, 23
readPolar(), 25
readSamsungHealth, 24
readShimmer, 25
readShimmer(), 14, 23
readTCX, 26
readTCX(), 15, 19
readWhoop, 27
readWithings, 28
readXsensMVNX, 29
readXsensMVNX(), 10, 12