Package 'EGM'

May 7, 2026

Title Intracardiac Electrograms

Version 0.2.0

Description A system for importing electrophysiological signal, based on the 'Waveform Database (WFDB)' software package, written by Moody et al 2022 <doi:10.13026/gjvw-1m31>. A R-based system to utilize 'WFDB' functions for reading and writing signal data, as well as functions for visualization and analysis are provided. A stable and broadly compatible class for working with signal data, supporting the reading in of cardiac electrophysiological files such as intracardiac electrograms, is introduced.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.2.0),

Imports vctrs (>= 0.5.0), data.table (>= 1.15.0), stats, fs, ggplot2, rlang, utils, xml2, base64enc, checkmate, stringr, signal,

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), withr, fastICA, moments, lifecycle,

LinkingTo cpp11

Config/testthat/edition 3

URL https://shah-in-boots.github.io/EGM/

BugReports https://github.com/shah-in-boots/EGM/issues

VignetteBuilder knitr

NeedsCompilation yes

Author Anish S. Shah [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-9729-1558>), Darren Seaney [ctb]

Maintainer Anish S. Shah <shah.in.boots@gmail.com>

Repository CRAN

Date/Publication 2025-11-02 06:11:13 UTC

Contents

`add_annotations` *Add annotations to a* `ggm` *object*

Description

The `add_annotations()` adds annotations to a `ggm` object. It is specific to this class as it requires the output of `ggm()` to included data stored in `annotation_table()`.

Usage

```
add_annotations(...)
```

Arguments

...	Arguments passed on to ggm
	data Data of the egm class, which includes header (meta) and signal information together.
	channels A character vector of which channels to use. Can give either the channel label (e.g "CS 1-2") or the recording device/catheter type (e.g "His" or "ECG"). If no channels are selected, the default is all channels.
	time_frame A time range that should be displaced given in the format of a vector with a length of 2. The left value is the start, and right value is the end time. This is given in seconds (decimals may be used).
	palette A character choice from the below options that describe the color choices to be used for plotting. If set to the default, which is NULL, no changes to the colors for individual channels will be performed. If a positive choice is made, then the background **mode** argument will be set to *dark* as the default, unless otherwise specified. *WARNING*: This is an experimental argument, and may be moved in future version. • **NULL**: no changes to the colors will be made. DEFAULT. • **material**: a colorscheme based off of the Material Design color scheme
	mode A character string from c("dark", "light") to describe the base/background color settings to be used. If there are preset channel colors that were exported in the egm object, these colors will be used for the individual channels. If **palette** is specified, then the *dark* option will be set automatically (a palette choice cannot be made without understanding the background to plate it across). *WARNING*: This is an experimental argument, and may be moved in future version. • The *dark* theme mimics the "white on black" scheme seen in *LabSystem Pro* format (and most other high-contrast visualizations), for minimizing eye strain. This calls the theme_egm_dark() function. DEFAULT. • The *light* theme mimics the "black on white" colors seen in the *Prucka* system. • NULL removes any theme, and uses the default ggplot2::ggplot() settings

---

add_colors *Add color scheme to a* ggm *object*

Description

Using add_colors() is part of the theme process for a ggm object, which in turn is a visual representation of an egm object. Often, the egm dataset will contain default colors based on where the signal data was brought in from. add_colors() can allow customization of those features to some degree based on *opinionated* color palettes.

Usage

```
add_colors(object, palette, mode)
```

Arguments

`object`	A `ggm` object
`palette`	A `character` choice from the below options that describe the color choices to be used for plotting. If set to the default, which is `NULL`, no changes to the colors for individual channels will be performed. If a positive choice is made, then the background **mode** argument will be set to *dark* as the default, unless otherwise specified. *WARNING*: This is an experimental argument, and may be moved in future version. • **NULL**: no changes to the colors will be made. DEFAULT. • **material**: a colorscheme based off of the Material Design color scheme
`mode`	A `character` string from `c("dark", "light")` to describe the base/background color settings to be used. If there are preset channel colors that were exported in the `egm` object, these colors will be used for the individual channels. If **palette** is specified, then the *dark* option will be set automatically (a palette choice cannot be made without understanding the background to plate it across). *WARNING*: This is an experimental argument, and may be moved in future version. • The *dark* theme mimics the "white on black" scheme seen in *LabSystem Pro* format (and most other high-contrast visualizations), for minimizing eye strain. This calls the `theme_egm_dark()` function. DEFAULT. • The *light* theme mimics the "black on white" colors seen in the *Prucka* system. • `NULL` removes any theme, and uses the default `ggplot2::ggplot()` settings

Details

Currently, the color choices are individual decided based on the channel source (e.g. lead) and are inspired by some modern palettes. The eventual goal for this function is to accept a multitude of palette options using heuristics similar to what is found in `{ggplot2}` or other graphing packages.

Value

Returns an updated `ggm` object

`analyze_atrial_signal` *Analyze F waves in atrial fibrillation ECG*

Description

Analyze F waves in atrial fibrillation ECG

Usage

```
analyze_atrial_signal(
  atrial_signal,
  frequency,
  characteristics = c("amplitude", "approximate_entropy", "dominant_frequency"),
  ...
)
```

Arguments

`atrial_signal`	Numeric vector of the atrial signal
`frequency`	Sampling frequency of the signal
`characteristics`	Vector of characteristics to analyze
`...`	Additional parameters for specific analyses

Value

A list containing the results of the requested analyses

`annotation_table` *Annotation Table*

Description

`annotation_table()` modifies the `data.table` class to work with annotation data. The columns are of all equal length, and each row describes a single annotation (although there may be duplicate time points).

Usage

```
annotation_table(
  annotator = character(),
  time = character(),
  sample = integer(),
  frequency = integer(),
  type = character(),
  subtype = character(),
  channel = integer(),
  number = integer(),
  ...
)

is_annotation_table(x)
```

Arguments

annotator	String that is the name of a WFDB-compatible annotation type, serving as the extension for the file that is written containing that annotation. Please see read_annotation() and write_annotation() for further details.
time	A character time stamp of the annotation, written in the format of **HH:MM:SS.SSS**, starting at **00:00:00.000**. This is converted to the appropriate time based on the header file (which records the actual start time and sampling frequency). This is often a missing variable and is given for compatibility with the WFDB applications.
sample	An integer representing the sample number of the annotation
frequency	An integer that represents the sampling frequency in Hertz
type	A character or string representing the type of the annotation
subtype	A character or string representing the subtype of the annotation
channel	An integer representing the channel number of the annotation, or a character representing the channel name
number	An additional integer value or number that classifies the annotation (allows for compatibility with multiple annotation types)
...	Additional arguments to be passed to the function
x	A data.table object that represents an annotation table

Details

The annotation_table() function creates a compatible table that can be used with write_annotation() and read_annotation() functions.

Value

A data.table that has invariant columns that are compatible with the WFDB library. The key columns include the sample index, the type of annotation (and its subtype and number qualifier), and the channel.

Annotation files

The following annotation file types are described below.

ecgpuwave**:**

ecgpuwave analyzes an ECG signal from the specified record, detecting the QRS complexes and locating the beginning, peak, and end of the P, QRS, and ST-T waveforms. The output of ecgpuwave is written as a standard WFDB-format annotation file (the extension is "*.ecgpuwave", as would be expected). This file can be converted into text format using rdann. Further details are given at the ECGPUWAVE page.

The **type** column can be *p*, *t*, or *N* for the peak of the P wave, T wave, and QRS (R peak) directly. The output notation also includes waveform onset XXX and waveform offset XXX. The **number** column gives further information about each of these **type** labels.

The **number** column gives modifier information. If the **type** classifier is a T wave annotation, the **number** column can be 0 (normal), 1 (inverted), 2 (positive), 3 (negative), 4 (biphasic negative-positive), 5 (biphasic positive-negative). If the **type** is an waveform onset or offset, then **number** can be 0 (P wave), 1 (QRS complex), 2 (T wave).

`annotators` *Annotator systems for WFDB objects*

Description

These functions create templates for annotation in R and extend the ability for developers to create their own annotation systems that are stable for WFDB objects. They are compatible with WFDB annotations and can be written out to a WFDB-compatible file. This also allows extensibility.

`as_ecg` *Convert an egm object to an ecg object*

Description

Converts a general `egm` object to a specialized `ecg` object for 12-lead ECG analysis.

Usage

```
as_ecg(x, ...)
```

Arguments

`x`	An object of class `egm`
`...`	Additional arguments

Value

An object of class `ecg`

`bard` *Read in ECG and EGM data from Bard (LabSystem Pro)*

Description

This function allows for reading in LS Pro data based on their text export of signals. Signals can be exported directly from the LS Pro system. The actual software is written by *Bard*.

The LabSystem Pro was acquired by Boston Scientific from the original company Bard. They are a common electrophysiology signal processing device for visualization and measurement of intracardiac signals.

Usage

```
read_bard(file, n = Inf)

read_bard_header(file)

read_bard_signal(file, n = Inf)
```

Arguments

`file`	The path to the file where the data is located. It must be a ***.txt** file. See details below about its format.
`n`	Number of signal values to return (this will be the same for each channel of data). Defaults to all values.

Value

An `egm` class object that is a list of EP signals the format of a `data.table`, with an attached **header** attribute that contains additional recording data.

Data Export

The steps to data export are as follows.

1. Start LabSystem PRO
2. Open a patient record
3. Display a waveform recording in a Review Window
4. Scroll to a point of interest in a waveform recording
5. Right click on the review window to the left of the region of interest
6. Select an Export option, either a default time range or the entire visible page (which depends on the sweep speed).

Data Format

```
[Header] Recording info - contains (example):

[Header]<CR><LF>
 File Type: 1<CR><LF>
 Version: 1<CR><LF>
 Channels exported: 22<CR><LF>
 Samples per channel: 5000<CR><LF>
 Start time:  6:55:24<CR><LF>
 End time:  6:55:29<CR><LF>
 Ch. Info. Pointer: 320<CR><LF>
 Stamp Data: T<CR><LF>
 Mux format: 0<CR><LF>
 Mux Block Size: <CR><LF>
 Data Format 1<CR><LF>
 Sample Rate: 1000Hz<CR><LF>

[Header] Channel info (per channel example):

  Channel #: 1<CR><LF>
  Label: III<CR><LF>
 Range: 5mv <CR><LF>
 Low: 1Hz<CR><LF>
 High: 100Hz<CR><LF>
 Sample rate: 1000Hz<CR><LF>
 Color: 0000FF<CR><LF>
 Scale: -7<CR><LF>

[Data] As described below:

-256,-1056,576,-256,320,-736,144,576,-592,176,608,240,176,-560,496,-
144,0,0,-32,-48,-32,-80<CR><LF>
```

Channel Data is interleaved in the example above (sample indexed at 1):

1	2	3	...	22
Ch1:1	Ch2:1	Ch3:1	...	Ch22:1
Ch1:2	Ch2:2	Ch3:2	...	Ch22:2
Ch1:3	Ch2:3	Ch3:3	...	Ch22:3
...	...	...	...	...
Ch1:5000	Ch2:5000	Ch3:5000	...	Ch22:5000

`c.windowed` *Concatenate windowed objects*

Description

Concatenate windowed objects

Usage

```
## S3 method for class 'windowed'
c(...)
```

Arguments

...	windowed objects to concatenate

Value

A windowed object containing all the elements of the input objects

---

calculate_approximate_entropy

Calculate Approximate Entropy (Ap_en) of a time series

Description

This function computes the approximate entropy (Ap_en) of a time series using the method described by Pincus (1991). Ap_en is a measure of the regularity and complexity of the time series. It is calculated by comparing vectors derived from the time series in an m-dimensional embedded space and in an (m+1)-dimensional space. The basic steps are:

1. **Embedding:** The time series is embedded into vectors of length m (and m+1) by taking successive elements. For a time series of length N, this produces (N - m + 1) (or (N - m) for m+1) vectors.
2. **Distance Calculation:** For each pair of embedded vectors, the Chebyshev distance (i.e., the maximum absolute difference among corresponding elements) is computed. If the distance between two vectors is less than or equal to a tolerance r, they are considered "similar."
3. **Counting and Averaging:** For each embedded vector, the function counts the number of similar vectors (including itself) and takes the natural logarithm of the ratio of this count to the total number of vectors. These log-values are then averaged to yield a statistic phi.
4. **Ap_en Calculation:** The approximate entropy is the difference between the phi computed for dimension m and the phi computed for dimension m+1, i.e., Ap_en = phi(m) - phi(m+1).

The tolerance r is typically chosen as a multiple of the standard deviation of the time series (commonly 3.5 * sd(x)). If r is not provided (or is negative), it is calculated automatically.

Usage

```
calculate_approximate_entropy(x, m = 3, r = NULL, implementation = "C++")
```

Arguments

x	Numeric vector of the time series
m	Embedding dimension (sample size), default is 3
r	Tolerance (threshold), default is 3.5 * sd(x)
implementation	Method to use for calculation, default is "C++", but can also be done in "R". The C++ implementation is faster.

Value

Approximate Entropy value

References

Pincus, S. M. (1991). Approximate entropy as a measure of system complexity. Proceedings of the National Academy of Sciences, 88(6), 2297-2301.

Examples

```
# Example: Calculate approximate entropy for a random time series
set.seed(123)
x <- rnorm(1000)
calculate_approximate_entropy(x, m = 3, r = -1, implementation = "R")
```

---

calculate_dominant_frequency

Calculate Dominant Frequency of a time series

Description

Calculate Dominant Frequency of a time series

Usage

```
calculate_dominant_frequency(x, frequency, f_min = 4, f_max = 9)
```

Arguments

x	Numeric vector of the time series
frequency	Sampling frequency of the signal
f_min	Minimum frequency to consider (default 4 Hz)
f_max	Maximum frequency to consider (default 9 Hz)

Value

Dominant Frequency in Hz

colors *Theming and color options for* ggm *objects*

Description

[Experimental]

The general purpose is to improve visualization of electrical signals. There is a pattern of colors that are generally given from different recording software, and they can be replicated to help improve visibility.

Usage

```
theme_egm()

theme_egm_light()

theme_egm_dark()
```

Value

A ggm object, with inheritance similar to `ggplot2::theme_minimal()`

color_channels *Identify the color for a channel based on palettes*

Description

This primarily restricts the colors to color-space safe options. It is intended to be used with `add_colors()` to provide a color scheme for the ggm object. It has been exposed to users for custom or advanced theming options.

Usage

```
color_channels(x, palette, mode = "dark")
```

Arguments

x	Vector of `character` names of requested ECG or EGM leads
palette	A `character` choice from the below options that describe the color choices to be used for plotting. If set to the default, which is `NULL`, no changes to the colors for individual channels will be performed. If a positive choice is made, then the background **mode** argument will be set to *dark* as the default, unless otherwise specified. *WARNING*: This is an experimental argument, and may be moved in future version.

- **NULL**: no changes to the colors will be made. DEFAULT.

- **material**: a colorscheme based off of the Material Design color scheme

`mode` A `character` string from `c("dark", "light")` to describe the base/background color settings to be used. If there are preset channel colors that were exported in the `egm` object, these colors will be used for the individual channels. If **palette** is specified, then the *dark* option will be set automatically (a palette choice cannot be made without understanding the background to plate it across). *WARNING*: This is an experimental argument, and may be moved in future version.

- The *dark* theme mimics the "white on black" scheme seen in *LabSystem Pro* format (and most other high-contrast visualizations), for minimizing eye strain. This calls the `theme_egm_dark()` function. DEFAULT.
- The *light* theme mimics the "black on white" colors seen in the *Prucka* system.
- `NULL` removes any theme, and uses the default `ggplot2::ggplot()` settings

Value

Vector of hex code colors as `character` based on the selected palette and light/dark mode

---

`detect_QRS` *Detect QRS complexes in ECG signals*

Description

`detect_QRS()` implements a modified Pan-Tompkins algorithm to detect QRS complexes in ECG signals. The function applies a sequence of processing steps including bandpass filtering, differentiation, squaring, and moving window integration to identify R peaks in the signal.

Usage

```
detect_QRS(signal, frequency, window_size = 0.15)
```

Arguments

`signal`	Numeric vector representing the ECG signal
`frequency`	Sampling frequency of the signal in Hz
`window_size`	Width of the integration window in seconds, default is 0.150 seconds

Details

The Pan-Tompkins algorithm is a widely-used method for QRS detection in ECG signals. This implementation follows these steps:

1. Bandpass filtering (5-15 Hz) to reduce noise and emphasize QRS complexes
2. Differentiation to highlight the steep slopes of QRS complexes 3. Squaring to amplify high-frequency components 4. Moving window integration to consider the overall QRS morphology 5. Adaptive thresholding to identify peaks 6. Application of a refractory period to prevent multiple detections of the same QRS complex

The function is designed to work with single-lead ECG signals, typically sampled at 250-1000 Hz.

Value

Integer vector containing the sample indices of detected QRS complexes

References

Pan, J., & Tompkins, W. J. (1985). A real-time QRS detection algorithm. IEEE Transactions on Biomedical Engineering, (3), 230-236. doi:10.1109/TBME.1985.325532

Examples

```
## Not run:
# Load ECG data
ecg_data <- read_muse(system.file("extdata", "muse-sinus.xml", package = "EGM"))

# Extract lead II signal
signal <- ecg_data$signal$II

# Get sampling frequency from header
freq <- attributes(ecg_data$header)$record_line$frequency

# Detect QRS complexes
qrs_locations <- detect_QRS(signal, freq)

# Plot ECG with detected QRS complexes
plot(signal, type = "l", xlab = "Sample", ylab = "Amplitude")
points(qrs_locations, signal[qrs_locations], col = "red", pch = 19)

## End(Not run)
```

---

ecg *Electrocardiogram data class for 12-lead ECG studies*

Description

This class serves as a specialized extension of the `egm` class specifically for standard 12-lead electrocardiogram data. It inherits all functionality from `egm` while providing additional validation and methods specific to 12-lead ECG analysis.

Usage

```
ecg(
  signal = signal_table(),
  header = header_table(),
  annotation = annotation_table(),
  ...
)

is_ecg(x)
```

Arguments

signal	A signal_table object generated by the signal_table() function containing standard ECG leads
header	A header_table object generated by the header_table() function
annotation	A annotation_table object generated by the annotation_table() function
...	Additional arguments to be passed to the function
x	An ecg object to be used with support functions

Details

The ecg object contains the same three components as egm:

- signal data in multiple channels (specifically 12 standard ECG leads)
- header information
- annotation labels at specified time points

The primary difference is that this class enforces validation to ensure the data represents a standard 12-lead ECG with appropriate lead names.

Value

An object of class ecg (which inherits from egm) containing signal, header, and annotation components.

---

egm *Electrogram data class from electrophysiology studies*

Description

This class serves as a combinatorial class to describe cardiovascular electrical signal data in R. It is based off of the formats available in WFDB, but has been formatted for ease of use within the R ecosystem. An egm object contains three components in a list:

- signal data in multiple channels
- header information
- annotation labels at specified time points

These components help to navigate, and visualize data. The egm class is the backbone for working with WFDB objects in R, and provides an interface for integrating or converting other raw signal data to a WFDB format.

Usage

```
egm(
  signal = signal_table(),
  header = header_table(),
  annotation = annotation_table(),
  ...
)

is_egm(x)
```

Arguments

signal	A signal_table object generated by the signal_table() function
header	A header_table object generated by the header_table() function
annotation	A annotation_table object generated by the annotation_table() function
...	Additional arguments to be passed to the function
x	An egm object, typically generated by the egm() function, to be used with support functions (e.g. is_egm()

Details

The individual components of the class are further defined in their respective children functions signal_table(), header_table(), annotation_table(). They are very simple classes that build upon the data.table class that allow for class safety checks when working with different data types (particularly WFDB).

IMPORTANT: The egm class can be built from ground-up by the user, however it is primarily generated for the user using the other read/write functions, such as read_bard() or read_wfdb().

Value

An object of class egm that is always a list of the above three components. Oftentimes, the annotation_table object may be missing, and it is replaced with an empty table as a place holder.

---

extract_f_waves *Extract F wave features from ECG*

Description

This function analyzes F waves in an ECG signal, extracting various characteristics.

Usage

```
extract_f_waves(
  object,
  lead = NULL,
  qrs_method = "adaptive_svd",
  f_characteristics = "amplitude",
  verbose = TRUE,
  .force_all = FALSE,
  ...
)
```

Arguments

object	An object of class egm or of subclass ecg
lead	Optional. A character string specifying the lead to analyze. If NULL (default), all available surface leads will be processed.
qrs_method	Method for ventricular signal removal. Default is "adaptive_svd" for adaptive singular value decomposition.
f_characteristics	Vector of characteristics to analyze from ECG signal. Options: "amplitude", "approximate_entropy", "dominant_frequency". Please see calculate_approximate_entropy() and calculate_dominant_frequency() for more details.
verbose	Logical. If TRUE, print information about which leads will be analyzed. Default is TRUE.
.force_all	Logical. If FALSE (default), only process surface ECG leads. If TRUE, process all available leads. This parameter is ignored if the object is of class 'ecg', in which case all leads are processed.
...	Additional arguments passed to methods

Value

A list containing F wave features for each processed lead

References

Park, Junbeom, Chungkeun Lee, Eran Leshem, Ira Blau, Sungsoo Kim, Jung Myung Lee, Jung-A Hwang, Byung-il Choi, Moon-Hyoung Lee, and Hye Jin Hwang. "Early Differentiation of Long-Standing Persistent Atrial Fibrillation Using the Characteristics of Fibrillatory Waves in Surface ECG Multi-Leads." Scientific Reports 9 (February 26, 2019): 2746. https://doi.org/10.1038/s41598-019-38928-6.

Hyvarinen, A., and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5), 411-430.

`extract_signal` *Extract raw signal data from an* `egm` *object*

Description

Raw signal data may be all that is required, particularly when storing or manipulating data, or for example, feeding it into an analytical pipeline. This means the extraneous elements, such as the *meta* information, may be unnecessary. This function helps to strip away and extract just the signal data itself and channel names.

Usage

```
extract_signal(object, data_format = c("data.frame", "matrix", "array"), ...)
```

Arguments

`object`	An `egm` object that contains the signal data to be extracted
`data_format`	A `character` choice of either *data.frame* (default), *matrix*, or *array* that tells how the data should be structured. Further explanation in the details.
`...`	Additional arguments to be passed to the function

Details

The options to return the data vary based on need. The data can be extracted as follows:

- `data.frame` containing an equal number of rows to the number of samples, with each column named after the recording channel it was derived from. Data frames, as they are columnar by nature, will also include the sample index position.
- `matrix` containing an equal number of rows to the number of samples, with each column named after the recording channel it was derived from
- `array` containing individual vectors of signal, each named after the channel they were derived from

Value

An object as described by the **format** option

format.windowed *Format a windowed object for printing*

Description

Format a windowed object for printing

Usage

```
## S3 method for class 'windowed'
format(x, ...)
```

Arguments

x	A windowed object
...	Additional arguments passed to methods

Value

Invisibly returns x

ggm *Visualization of EGMs using* ggplot

Description

[Experimental]

The ggm() function is used to plot objects of the egm class. This function however is more than just a plotting function - it serves as a visualization tool and confirmation of patterns, annotations, and underlying waveforms in the data. The power of this, instead of being a geom_*() object, is that annotations, intervals, and measurements can be added incrementally.

Usage

```
ggm(
  data,
  channels = character(),
  time_frame = NULL,
  palette = NULL,
  mode = "dark",
  ...
)
```

Arguments

`data`	Data of the `egm` class, which includes header (meta) and signal information together.
`channels`	A `character` vector of which channels to use. Can give either the channel label (e.g "CS 1-2") or the recording device/catheter type (e.g "His" or "ECG"). If no channels are selected, the default is all channels.
`time_frame`	A time range that should be displaced given in the format of a vector with a length of 2. The left value is the start, and right value is the end time. This is given in seconds (decimals may be used).
`palette`	A `character` choice from the below options that describe the color choices to be used for plotting. If set to the default, which is `NULL`, no changes to the colors for individual channels will be performed. If a positive choice is made, then the background **mode** argument will be set to *dark* as the default, unless otherwise specified. *WARNING*: This is an experimental argument, and may be moved in future version. • **NULL**: no changes to the colors will be made. DEFAULT. • **material**: a colorscheme based off of the Material Design color scheme
`mode`	A `character` string from `c("dark", "light")` to describe the base/background color settings to be used. If there are preset channel colors that were exported in the `egm` object, these colors will be used for the individual channels. If **palette** is specified, then the *dark* option will be set automatically (a palette choice cannot be made without understanding the background to plate it across). *WARNING*: This is an experimental argument, and may be moved in future version. • The *dark* theme mimics the "white on black" scheme seen in *LabSystem Pro* format (and most other high-contrast visualizations), for minimizing eye strain. This calls the `theme_egm_dark()` function. DEFAULT. • The *light* theme mimics the "black on white" colors seen in the *Prucka* system. • `NULL` removes any theme, and uses the default `ggplot2::ggplot()` settings
`...`	Additional arguments to be passed to the function

Value

An `{ggplot2}` compatible object with the `ggm` class, which contains additional elements about the header and annotations of the original data.

`header_table` *Header Table*

Description

`header_table()` modifies the `data.table` class to work with header data. The header data is read in from a similar format as to that of WFDB files and should be compatible/interchangeable when writing out to disk. The details extensively cover the type of data that is input. Generally, this function is called by `read_*_header()` functions and will generally not be called by the end-user.

Usage

```
header_table(
  record_name = character(),
  number_of_channels = integer(),
  frequency = 250,
  samples = integer(),
  start_time = strptime(Sys.time(), "%Y-%m-%d %H:%M:%OSn"),
  ADC_saturation = integer(),
  file_name = character(),
  storage_format = 16L,
  ADC_gain = 200L,
  ADC_baseline = ADC_zero,
  ADC_units = "mV",
  ADC_resolution = 12L,
  ADC_zero = 0L,
  initial_value = ADC_zero,
  checksum = 0L,
  blocksize = 0L,
  label = character(),
  info_strings = list(),
  additional_gain = 1,
  low_pass = integer(),
  high_pass = integer(),
  color = "#000000",
  scale = integer()
)

is_header_table(x)
```

Arguments

record_name	A character vector of record line information
number_of_channels	An integer describing number of signals
frequency	A numeric value of sampling frequency, 250 Hz default
samples	An integer for the number of samples
start_time	The POSIXct time of recording, with miliseconds included. For example, strptime(start_time, "%Y-%m-%d %H:%M%:%OSn") where as described in base::strptime()
ADC_saturation	An integer representing ADC saturation
file_name	A character for the signal specific information
storage_format	An integer of the bits for the storage format, 16-bit default
ADC_gain	An integer of ADC gain, default of 200
ADC_baseline	An integer of ADC baseline, defaults to **ADC_zero**
ADC_units	A character to describe ADC units, "mV" is default
ADC_resolution	An integer for ADC resolution, default is 12

ADC_zero	An integer for ADC zero, defaults to 0
initial_value	An integer for the initial value, defaults to **ADC_zero** value
checksum	An integer that serves as the checksum
blocksize	An integer of the block size
label	A character description of the signal
info_strings	A list of strings that will be written as an appendix to the header file, usually containing information about the channels, (e.g. list of colors, extra labels, etc).
additional_gain	A numeric Additional gain, defaults to 1.0
low_pass	An integer Low pass filter
high_pass	An integer High pass filter
color	A character Color as hexadecimal format, defaults to black
scale	An integer Scale
x	A data.table object that serves as the header table

Details

The header_table object is relatively complex in that it directly deals with properties of the signal, and allows compatibility with WFDB files and other raw header files for other signal objects. It can be written out using write_wfdb().

Value

A header_table object that is an extension of the data.table class. This contains an adaptation of the function arguments, allowing for compatibility with the WFDB class.

Header file structure

There are three components to the header file:

1. **Record line** that contains the following information, in the order documented, however pieces may be missing based on different parameters. From left to right...
   - Record name
   - Number of signals: represents number of segments/channels
   - Sampling frequency (optional)
   - Number of samples (optional)
   - Time: in HH:MM:SS format (optional)
   - Date: in DD/MM/YYYY (optional)
2. **Signal specification lines** contains specifications for individual signals, and there must be as many signal lines as there are reported by the above record line. From left to right....
   - File name: usually *.dat
   - Format integer: represents storage type, e.g. 8-bit or 16-bit
   - ADC gain: ADC units per physical unit (optional)
     - Baseline: corresponds to 0 physical units, sep = '*(0)" (optional)

  - Units: with '/' as a field separator e.g '*/mV' (optional)
- ADC resolution `integer`: bits, usually 8 or 16 (optional)
- ADC zero: represents middle of ADC input range (optional)
- Initial value (optional)
- Checksum (optional)
- Block size (optional)
- Description: text or label information (optional)

3. **Info strings** are unstructured lines that contains information about the record. Usually are descriptive. Starts with initial '#' without preceding white space at beginning of line.

`is_windowed` *Test if an object is a windowed object*

Description

Test if an object is a windowed object

Usage

```
is_windowed(x)
```

Arguments

`x`	An object to test

Value

TRUE if x is a windowed object, FALSE otherwise

`lapply.windowed` *Apply a function to each element of a windowed object*

Description

Apply a function to each element of a windowed object

Usage

```
lapply.windowed(X, FUN, ...)
```

Arguments

`X`	A windowed object
`FUN`	A function to apply to each element
`...`	Additional arguments passed to FUN

Value

A list of the results of applying FUN to each element of X, or a new windowed object if all results are egm objects

---

muse *Read in ECG data from MUSE*

Description

This function serves to read/convert XML based files from the MUSE system to digital signal. This can subsequently be written into other formats. The MUSE system is somewhat proprietary, and each version may or may not allow export options into XML.

Usage

```
read_muse(file)
```

Arguments

file	An ECG file from MUSE in XML format

Details

GE Healthcare MUSE v9 is currently the model that is being used. These functions have not been tested in older versions.

Value

An egm class object that is a list of eps signals the format of a data.table, with an attached **header** attribute that contains additional recording data.

---

print.windowed *Print a windowed object*

Description

Print a windowed object

Usage

```
## S3 method for class 'windowed'
print(x, ...)
```

Arguments

x	A windowed object
...	Additional arguments passed to methods

Value

Invisibly returns x

`prucka` *Read Prucka System Files*

Description

`read_prucka()` reads both the signal data (*.txt) and header information (*.inf) exported from the Prucka cardiac electrophysiology system, which is the underlying recording software used in GE Healthcare's CardioLab EP system.

Usage

```
read_prucka(signal_file, header_file = NULL, n = Inf)

read_prucka_header(header_file)

read_prucka_signal(signal_file, n = Inf)
```

Arguments

`signal_file`	Path to the *.txt signal data file
`header_file`	Path to the *.inf header file. If NULL, will look for a file with the same base name as signal_file but with .inf extension.
`n`	Number of signal values to return (this will be the same for each channel of data). Defaults to all values.

Details

Exporting from GE CardioLab/Prucka:

To export data from the GE CardioLab system:

1. Open the study/recording in CardioLab
2. Select the time segment you want to export
3. Navigate to **File > Export** or **Tools > Export**
4. Choose **ASCII Export** or **Text Export** format
5. Select the channels to export
6. Choose export location and filename
7. The system will create two files:
   - **X####.txt**: Space-delimited signal data

- **X####.inf**: Header file with metadata

File Format Details:

Signal file (*.txt):

- Space-delimited numeric data
- Each row represents one time point
- First column: sample index/time marker
- Subsequent columns: channel data in mV
- All channels sampled at the same rate

Header file (*.inf):

- Key-value pairs with "=" delimiter
- Patient information (name, date, description)
- Recording parameters (sampling rate, duration, channel count)
- Channel mapping section listing channel numbers and labels
- Channel numbers may be non-sequential (e.g., 1-12, 49-50, 75-76)

Both files must have the same base name (e.g., X001.txt and X001.inf).

Notes:

- Default units are mV for electrical signals and mmHg for pressure
- The system typically uses 16-bit ADC resolution
- Channel labels may include surface ECG leads (I, II, III, aVR, aVL, aVF, V1-V6) and intracardiac catheters (ABL, His, CS, RV, etc.)
- Export may be limited by system memory for very long recordings

Value

An `egm` class object that is a list of EP signals the format of a `data.table`, with an attached **header** attribute that contains additional recording data.

`signal_table` *Signal tables*

Description

The `signal_table()` function modifies the `data.table` class to work with electrical signal data. The input should be a data set of equal number of rows. It will add a column of index positions called `sample` if it does not already exist.

Usage

```
signal_table(...)

is_signal_table(x)
```

Arguments

...	A list of equal lengths
x	data.frame A data frame of signal data

Value

An object of class signal_table, which is an extension of the data.table class. The sample column is *invariant* and will always be present. The other columns represent additional channels.

---

standardize_windows *Standardize windows of signal data*

Description

Standardizes windowed objects by applying various transformations to each window. This function converts each egm object in a windowed list to a standardized data frame with uniform properties, facilitating comparison and analysis.

Usage

```
standardize_windows(
  x,
  standardization_method = c("time_normalize"),
  target_samples = 500,
  target_ms = NULL,
  interpolation_method = c("linear", "spline", "step"),
  align_feature = NULL,
  preserve_amplitude = TRUE,
  preserve_class = FALSE,
  ...
)
```

Arguments

x	A windowed object to standardize
standardization_method	A character string specifying the standardization method. Currently supported: "time_normalize".
target_samples	The desired number of samples for each standardized window. Default is 500 samples. This parameter takes precedence if both target_samples and target_ms are provided.
target_ms	Alternative specification in milliseconds. If provided and target_samples is NULL, the function will convert this to samples based on the signal's sampling frequency.

`interpolation_method`
: The method used for interpolation when resampling. Options are "linear" (default), "spline", or "step".

`align_feature`
: Feature to align windows around, either a character string matching an annotation type or a list of criteria for finding a specific annotation. Default is NULL (no alignment).

`preserve_amplitude`
: Logical. If TRUE (default), maintains original amplitude range after resampling.

`preserve_class`
: Logical. If TRUE, returns a `windowed` object with standardized data frames. If FALSE (default), returns a plain list of data frames.

`...`
: Additional arguments passed to specific standardization methods.

Details

Currently supported standardization methods:

- `time_normalize` - Resamples each window to a standard length by either dilating or contracting the signal. The result is a signal with a consistent number of samples regardless of the original window duration.

Additional options:

- `align_feature` - If provided, windows will be aligned to center around this feature (e.g., a specific annotation type like "N" for R-peak). Can be a character string matching an annotation type or a list of criteria for annotation matching.
- `preserve_amplitude` - If TRUE (default), maintains the original amplitude range after resampling. If FALSE, the amplitudes may change due to interpolation.

Value

If `preserve_class=TRUE`, a `windowed` object containing standardized data frames. If `preserve_class=FALSE`, a plain list of standardized data frames.

Examples

```
## Not run:
# Read in ECG data
ecg <- read_wfdb("ecg", test_path(), "ecgpuwave")

# Create windows based on sinus rhythm
windows <- window_signal(
  ecg,
  method = "rhythm",
  rhythm_type = "sinus",
  onset_criteria = list(type = "(", number = 0),
  offset_criteria = list(type = ")", number = 2),
  reference_criteria = list(type = "N")
)

# Standardize windows to exactly 500 samples
```

```
std_windows <- standardize_windows(
  windows,
  method = "time_normalize",
  target_samples = 500
)

# Alternatively, standardize to 500 milliseconds (depends on sampling frequency)
std_windows_ms <- standardize_windows(
  windows,
  method = "time_normalize",
  target_ms = 500
)

# Standardize windows with QRS alignment
aligned_windows <- standardize_windows(
  windows,
  method = "time_normalize",
  target_samples = 500,
  align_feature = "N"  # Align on QRS complexes
)

## End(Not run)
```

---

wfdb *Waveform Database (WFDB) Software Package*

Description

This implementation of WFDB is a back-end for the WFDB using a combination of *python*, *C++*, and *C* language. The related functions are documented separately. This serves as an overview of the conversion of WFDB formats to R formats. In this documentation, the specific WFDB generated files will be described.

Arguments

record	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
record_dir	File path of directory that should be used read and write files. Defaults to current directory.
annotator	String that is the name of a WFDB-compatible annotation type, serving as the extension for the file that is written containing that annotation. Please see read_annotation() and write_annotation() for further details.
...	Additional arguments to be passed to the function

WFDB

The WFDB (Waveform Database) Software Package has been developed over the past thirty years, providing a large collection of software for processing and analyzing physiological waveforms. The package is written in highly portable C and can be used on all popular platforms, including GNU/Linux, MacOS X, MS-Windows, and all versions of Unix.

The foundation of the WFDB Software Package is the WFDB library, consisting of a set of functions for reading and writing digitized signals and annotations. These functions can be used by programs written in C, C++, or Fortran, running under any operating system for which an ANSI/ISO C compiler is available, including all versions of Unix, MS-DOS, MS-Windows, the Macintosh OS, and VMS.

Data format

The records that the WFDB uses have three components...

1. Signals: integer values that are at equal intervals at a certain sampling frequency
2. Header attributes: recording information such as sample number, gain, sampling frequency
3. Annotations: information about the record such as a beat labels or alarm triggers

Author(s)

Original software: George Moody, Tom Pollard, Benjamin Moody
R implementation: Anish S. Shah
Last updated: 2025-11-01

`wfdb_annotations` *Read WFDB-compatible annotation file*

Description

Individual annotation types are described as both a command-line tool for annotating WFDB-files, as well as the extension that is appended to the `record` name to notate the type. Generally, the types of annotations that are supported are described below:

- atr = manually reviewed and corrected reference annotation files
- ann = general annotator file
- ecgpuwave = files contain surface ECG demarcation (P, QRS, and T waves)
- sqrs/wqrs/gqrs = standard WFDB peak detection for R waves

A more thorough explanation is given in the details. Additionally, files when being read in are converted from a binary format to a textual format. The raw data however may be inadequate, as the original annotation may be erroneous. In these cases, an empty `annotation_table` object will be returned.

Usage

```
read_annotation(
  record,
  annotator,
  record_dir = ".",
  begin = 0,
  end = NA_real_,
  header = NULL
)

write_annotation(data, annotator, record, record_dir = ".")
```

Arguments

record	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
annotator	String that is the name of a WFDB-compatible annotation type, serving as the extension for the file that is written containing that annotation. Please see `read_annotation()` and `write_annotation()` for further details.
record_dir	File path of directory that should be used read and write files. Defaults to current directory.
begin, end	A `character` in the format of *HH:MM:SS* that will be used to help parse the time of the annotation. These parameters together create the time range to extract. The default of *0* is a shortcut for *00:00:00*. The *seconds* argument can include a decimal place.
header	A header file is an optional named list of parameters that will be used to organize and describe the signal input from the **data** argument. If the **type** is given, specific additional elements will be searched for, such as the low or high pass filters, colors, or other signal attributes. At minimum, the following elements are required (as cannot be calculated): • frequency = sample frequency in Hertz as `integer` • label = vector of names for each channel as `character` • start_time = date/time object
data	An `annotation_table` containing the 6 invariant columns required by the `annotation_table()` function

Value

This function will either read in an annotation using the `read_annotation()` function in the format of an `annotation_table` object, or write to file/disk an `annotation_table` to a WFDB-compatible annotation file using the `write_annotation()` function.

IMPORTANT: as annotation files are created by annotators that were developed independently, there is a higher chance of an erroroneous file being created on disk. As such, this function will note an error an return an empty `annotation_table` at times.

Annotation files

The following annotation file types are described below.

`ecgpuwave`**:**

`ecgpuwave` analyzes an ECG signal from the specified record, detecting the QRS complexes and locating the beginning, peak, and end of the P, QRS, and ST-T waveforms. The output of ecgpuwave is written as a standard WFDB-format annotation file (the extension is "*.ecgpuwave", as would be expected). This file can be converted into text format using `rdann`. Further details are given at the ECGPUWAVE page.

The **type** column can be *p*, *t*, or *N* for the peak of the P wave, T wave, and QRS (R peak) directly. The output notation also includes waveform onset XXX and waveform offset XXX. The **number** column gives further information about each of these **type** labels.

The **number** column gives modifier information. If the **type** classifier is a T wave annotation, the **number** column can be 0 (normal), 1 (inverted), 2 (positive), 3 (negative), 4 (biphasic negative-positive), 5 (biphasic positive-negative). If the **type** is an waveform onset or offset, then **number** can be 0 (P wave), 1 (QRS complex), 2 (T wave).

`wfdb_annotation_labels` *Standard WFDB annotation nomenclature*

Description

Provides the standard label definitions used by WFDB annotation files. These helper functions make it easier to interpret the contents of the `annotation_table()` object by exposing the symbol, mnemonic, and description that correspond to each label store value defined in the WFDB Applications Guide (Moody and collaborators).

Usage

```
wfdb_annotation_labels(symbol = NULL, label_store = NULL)

wfdb_annotation_decode(annotation, column = "type")
```

Arguments

`symbol`	Optional character vector of WFDB annotation symbols to filter the results.
`label_store`	Optional integer vector of WFDB label store values to filter the results.
`annotation`	An `annotation_table()` or compatible data frame whose annotation symbols or label store values should be augmented with the standard WFDB nomenclature.
`column`	Name of the column within `annotation` that contains either the WFDB symbol (default, for the `type` column) or the label store values. If the column is numeric it is matched on `label_store`, otherwise a symbol lookup is performed.

Details

The returned table is derived from the WFDB Application Guide and matches the canonical label store values used by the WFDB software distribution. Entries that are not currently defined by the specification are omitted.

Value

`wfdb_annotation_labels()` returns a data frame with columns `label_store`, `symbol`, `mnemonic`, and `description`. `wfdb_annotation_decode()` returns the input annotation table with the WFDB nomenclature columns appended.

References

Moody GB. *WFDB Applications Guide*. PhysioNet. Available at https://www.physionet.org/physiotools/wag/.

Examples

```
wfdb_annotation_labels()

wfdb_annotation_labels(symbol = c("N", "V"))

ann <- annotation_table(
  annotator = "example",
  sample = c(100L, 200L),
  type = c("N", "V")
)

wfdb_annotation_decode(ann)
```

`wfdb_io` *I/O of WFDB-compatible signal & header files from EP recording systems*

Description

This function allows for WFDB files to be read from any WFDB-compatible system, and also allows writing out WFDB-compatible files from specific EP recording systems, as indicated in the details section. Writing WFDB leads to creation of both a **dat** (signal) and **hea** (header) file. These are both required for reading in files as well.

Usage

```
write_wfdb(
  data,
  record,
  record_dir = ".",
```

```
  header = NULL,
  info_strings = list(),
  units = c("digital", "physical"),
  ...
)

read_wfdb(
  record,
  record_dir = ".",
  annotator = NULL,
  begin = 0,
  end = NA_integer_,
  interval = NA_integer_,
  units = c("digital", "physical"),
  channels = character(),
  ...
)

read_signal(
  record,
  record_dir = ".",
  header = NULL,
  begin = 0,
  end = NA_integer_,
  interval = NA_integer_,
  units = c("digital", "physical"),
  channels = character(),
  ...
)

read_header(record, record_dir = ".", ...)
```

Arguments

data	Can either be an egm object, or a data.frame (or similar) object. The function will appropriately set defaults based on the type. • egm = Will extract signal and header data directly from object, and thus is simplest to convert to a WFDB format • signal_table = This is a customized data.table class that has an invariant column containing sample information. • data.frame or data.table = Must have a column that represents a time point or index, and columns that represent signal values (preferably integers)
record	String that will be used to name the WFDB record. Cannot include extensions, and is not a filepath. alphanumeric characters are acceptable, as well as hyphens (-) and underscores (_)
record_dir	File path of directory that should be used read and write files. Defaults to current directory.

`header` A header file is an optional named list of parameters that will be used to organize and describe the signal input from the **data** argument. If the **type** is given, specific additional elements will be searched for, such as the low or high pass filters, colors, or other signal attributes. At minimum, the following elements are required (as cannot be calculated):

- frequency = sample frequency in Hertz as `integer`
- label = vector of names for each channel as `character`
- start_time = date/time object

`info_strings` A `list` of strings that will be written as an appendix to the header file, usually containing information about the channels, (e.g. list of colors, extra labels, etc).

`units` A `character` string representing either *digital* (DEFAULT) or *physical* units that should be used for signal values.

- `"digital"` = Returns raw ADC (analog-to-digital converter) counts as stored in the .dat file. These are integer values representing the digitized signal without any scaling applied. Use this to preserve exact round-trip fidelity.
- `"physical"` = Returns signal values converted to physical units (e.g., mV) using the formula: `physical = (digital - baseline) / gain`, where `baseline` and `gain` are specified in the header file. This is the human-readable format for analysis.

`...` Additional arguments to be passed to the function

`annotator` String that is the name of a WFDB-compatible annotation type, serving as the extension for the file that is written containing that annotation. Please see `read_annotation()` and `write_annotation()` for further details.

`begin, end, interval`
Timepoint as an `integer` (representing seconds), which is converted to an index position based on sampling frequency. The default is to start at the beginning of the record. If `end` or `interval` are given, the earlier of the two will be returned. The `end` argument gives a time index to read until. The `interval` argument is the length of time past the start point.

`channels` Either the signal/channel in a `character` vector as a name or number. Allows for duplication of signal or to re-order signal if needed. If nothing is given, will default to all channels available.

Details

The `begin`, `end`, and `interval` arguments are converted into sample positions using the sampling frequency declared in the WFDB header. The reader first determines the starting sample from `begin`, then gives precedence to `interval` (when supplied) before falling back to `end`. Any request that extends beyond the recorded range is clamped so that the caller still receives all available data without a hard failure.

Value

Depends on if it is a reading or writing function. For writing, will output an WFDB-based object reflecting the function. For reading, will output an extension of a `data.table` object reflecting the underlying function (e.g. `signal_table()` will return an object of class).

Functions

- `write_wfdb()`: Writes out signal and header data into a WFDB-compatible format from R. The `units` parameter indicates whether the input signal data is in digital (raw ADC counts) or physical units. When `units="physical"`, the function automatically converts to digital units using the inverse formula: `digital = (physical * gain) + baseline` before writing to disk.
- `read_wfdb()`: Reads a multicomponent WFDB-formatted set of files directly into an `egm` object. This serves to pull together `read_signal()`, `read_header()`, and `read_annotation()` for simplicity.
- `read_signal()`: Specifically reads the signal data from the WFDB binary format, returning a `signal_table` object for evaluation in the R environment
- `read_header()`: Specifically reads the header data from the WFDB header text format, returning a `header_table` object for evaluation in the R environment

Recording systems

Type of signal data, as specified by the recording system, that are currently supported.

- *bard* = Bard (LabSystem Pro), e.g. `read_bard()`
- *muse* = MUSE (GE), e.g. `read_muse()`
- *prucka* = Prucka (CardioLab), e.g. `read_prucka()`

`wfdb_paths` *WFDB path utilities*

Description

These functions are used to help find and locate commands from the installation of WFDB. They are helpful in setting and getting path options and specific WFDB commands. They are primarily internal helper functions, but are documented for troubleshooting purposes.

Usage

```
find_wfdb_software()

set_wfdb_path(.path)

find_wfdb_command(.app, .path = getOption("wfdb_path"))
```

Arguments

`.path`	A `character` string that describes the path to the WFDB binary directory
`.app`	The name of WFDB software command or application as a `character`

Value

These functions are helper functions to work with the user-installed WFDB software. They do not always return an object, and are primarily used for their side effects. They are primarily developer functions, but are exposed to the user to help troubleshoot issues with their installation of WFDB.

`window` *Window signal data based on different methods*

Description

[Experimental]

Creates windows of signal data using various methods, such as rhythm patterns, time intervals, or reference points. Each window is returned as an individual `egm` object for further analysis.

Usage

```
window(object, window_method = c("rhythm"), ...)

window_by_rhythm(
  object,
  rhythm_type = "sinus",
  onset_criteria,
  offset_criteria,
  reference_criteria = NULL,
  adjust_sample_indices = TRUE,
  ...
)
```

Arguments

`object`	Object of the `egm` class, which includes header, signal information, and annotation information.
`window_method`	A `character` string specifying the windowing method. Options include: • **rhythm** - Windows based on rhythm patterns (requires rhythm_type and criteria)
`...`	Additional arguments passed to specific windowing methods.
`rhythm_type`	A `character` string specifying the rhythm type (e.g., "sinus"). Currently supported: "sinus" (requires reference check).
`onset_criteria`	A named list of criteria to identify onset points. Names should match column names in the annotation table.
`offset_criteria`	A named list of criteria to identify offset points. Names should match column names in the annotation table.

reference_criteria
: A named list of criteria to identify reference points that must exist between onset and offset. Set to NULL to skip reference validation.

adjust_sample_indices
: Logical, whether to adjust annotation sample indices in the returned windows to be relative to the window start. Default is TRUE.

Details

This function provides a modular approach to windowing electrophysiological signals. The method parameter determines the windowing strategy, with each method requiring its own set of additional parameters.

Value

A list of egm objects, each representing a window of the original signal.

---

windowed *Create a* windowed *object containing a list of egm segments*

Description

[Experimental]

windowed objects are lists of egm objects that represent segments or windows of the original signal. This allows for specialized methods to be applied to collections of signal windows. This function primarily serves as the class generation function, and only applies class attributes. It is used by the window() function to ensure appropriate class and properties.

Usage

```
windowed(
  x = list(),
  window_method = "rhythm",
  source_record = character(),
  ...
)
```

Arguments

x	A list of egm objects
window_method	The windowing method used to create the list
source_record	The name of the original record
...	Additional arguments passed to methods

Value

An object of class windowed which inherits from list

`[.windowed` *Subset a windowed object*

Description

Subset a windowed object

Usage

```
## S3 method for class 'windowed'
x[i, ...]
```

Arguments

`x`	A windowed object
`i`	Index to subset
`...`	Additional arguments passed to methods

Value

A windowed object with the specified subset of elements

Index