

# Package ‘mnirs’

March 30, 2026

**Title** Muscle Near-Infrared Spectroscopy Processing and Analysis

**Version** 0.6.0

**Description** Read, process, and analyse data from muscle near-infrared spectroscopy (mNIRS) devices. Import raw data from .csv or .xls(x) files and return time-series data and metadata. Includes standardised methods for cleaning, filtering, and pre-processing mNIRS data for subsequent analysis. Also includes a custom plot theme and colour palette. Intended for mNIRS researchers and practitioners in exercise physiology, sports science, and clinical rehabilitation with minimal coding experience required.

**License** MIT + file LICENSE

**URL** <https://jemarnold.github.io/mnirs/>,  
<https://github.com/jemarnold/mnirs>

**BugReports** <https://github.com/jemarnold/mnirs/issues>

**Depends** R (>= 4.1)

**Imports** cli, data.table, lifecycle, readxl, rlang, stats, tibble,  
tidyselect

**Suggests** dplyr, ggplot2, knitr, patchwork, quarto, scales, signal,  
testthat (>= 3.0.0), zoo

**VignetteBuilder** quarto

**Config/Needs/website** quarto, tidyverse

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Jem Arnold [aut, cre, cph] (ORCID:  
<https://orcid.org/0000-0003-3908-9447>>)

**Maintainer** Jem Arnold <jem.arnold@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-03-30 08:40:02 UTC

## Contents

artinis_intervals.xlsx	2
breaks_timespan	3
by_time	4
create_mnirs_data	5
example_mnirs	7
extract_intervals	7
filter_butter	11
filter_ma	13
filter_mnirs	15
format_hmmss	18
moxy_intervals.csv	19
moxy_ramp.xlsx	20
palette_mnirs	21
plot.mnirs	22
portamon-oxcap.xlsx	23
read_mnirs	24
replace_mnirs	26
resample_mnirs	31
rescale_mnirs	33
scale_colour_mnirs	35
shift_mnirs	36
theme_mnirs	38
train.red_intervals.csv	39
<b>Index</b>	<b>41</b>

---

artinis\_intervals.xlsx

*10 Hz Artinis Oxysoft export recorded with Oxymon MKIII*

---

### Description

Exported from Artinis Oxysoft, recorded on Oxymon MKIII at 50 Hz and exported at 10 Hz. Containing two 5-minute cycling work intervals and an ischaemic occlusion, placed on the vastus lateralis muscle site.

### Format

.xlsx file with header metadata and five columns and 20919 rows:

**Column 1** Sample index (divide by sample rate for seconds).

**Column 2** O2Hb: oxyhaemoglobin concentration change ( $\mu\text{M}$ ).

**Column 3** HHb: deoxyhaemoglobin concentration change ( $\mu\text{M}$ ).

**Column 4** Event marker (character).

**Column 5** Unmarked event label (character).

Channel mapping for `read_mnirs()`:

- `nirs_channels = c(O2Hb = 2, HHb = 3)`
- `time_channel = c(sample = 1)`
- `event_channel = c(event = 4, label = "col_5")`
- `interval_times = list(start = c(158, 999, 1750) end = c(493, 1333, 1961))`  
two intervals and post-exercise occlusion

### Source

Artinis Medical Systems. Oxymon MKIII, exported via Oxysoft desktop software (<https://www.artinis.com/>)

### See Also

`read_mnirs()`, `example_mnirs()`

### Examples

```
example_mnirs("artinis_intervals")
```

---

breaks_timespan	<i>Breaks for timespan data</i>
-----------------	---------------------------------

---

### Description

Pretty timespan breaks for plotting in units of 5, 15, 30, 60 sec, etc. Modified from `scales::breaks_timespan()`.

### Usage

```
breaks_timespan(unit = c("secs", "mins", "hours", "days", "weeks"), n = 5)
```

### Arguments

- |                   |  |
|-------------------|--|
| <code>unit</code> | The time unit used to interpret numeric data input ( <i>defaults</i> to <i>"secs"</i> ). |
| <code>n</code>    | Desired number of breaks. You may get slightly more or fewer breaks than requested.      |

### Value

Returns a function for generating breaks.

**Examples**

```
x <- 0:120
y <- sin(2 * pi * x / 15) + rnorm(length(x), 0, 0.2)

ggplot2::ggplot(data.frame(x, y), ggplot2::aes(x, y)) +
  theme_mnirs() +
  ggplot2::scale_x_continuous(breaks = breaks_timespan()) +
  ggplot2::geom_line()
```

---

by_time	<i>Specify interval boundaries by time, label, lap, or sample</i>
---------	---

---

**Description**

Helper functions to define interval start or end boundaries for `extract_intervals()`.

**Usage**

```
by_time(...)
by_sample(...)
by_label(...)
by_lap(...)
```

**Arguments**

... Specify start or end boundaries.

by\_time(...) Numeric time values in units of time\_channel.

by\_label(...) Character strings to match in event\_channel. All matching occurrences are returned.

by\_lap(...) Integer lap numbers to match in event\_channel. For start, resolves to the first sample of each lap. For end, resolves to the last sample.

by\_sample(...) Integer sample indices (row numbers).

**Details**

These helpers can be used explicitly for arguments start/end, or raw values can be passed directly:

- Numeric → `by_time()`
- Character → `by_label()`,
- Explicit integer (e.g. 2L) → `by_lap()`.
- Use `by_sample()` explicitly for sample indices.

**Value**

An object of class "mnirs\_interval" for use with the start and end arguments of `extract_intervals()`.

**Examples**

```
## read example data
data <- read_mnirs(
  example_mnirs("train.red"),
  nirs_channels = c(
    smo2_left = "SmO2 unfiltered",
    smo2_right = "SmO2 unfiltered"
  ),
  time_channel = c(time = "Timestamp (seconds passed)"),
  event_channel = c(lap = "Lap/Event"),
  zero_time = TRUE,
  verbose = FALSE
)

## start and end by time
extract_intervals(data, start = by_time(66), end = by_time(357))

## start by lap
extract_intervals(data, start = by_lap(2, 4), span = 0)

## introduce event_channel with "start" string
data$event <- NA_character_
data$event[1000] <- "start"
data <- create_mnirs_data(data, event_channel = "event")

## start by label, end by time
extract_intervals(data, start = by_label("start"), end = by_time(1500))

## multiple intervals by sample index
extract_intervals(data, start = by_sample(1000, 1500), end = by_sample(2000, 2600))
```

---

create_mnirs_data	<i>Create an mnirs data frame with metadata</i>
-------------------	---

---

**Description**

Manually add class "mnirs" and metadata to an existing data frame.

**Usage**

```
create_mnirs_data(data, ...)
```

## Arguments

data	A data frame with existing metadata (accessed with <code>attributes(data)</code> ).
...	Additional arguments with metadata to add to the data frame. Can be either separate named arguments or a list of named values. <ul style="list-style-type: none"><li>• nirs_device</li><li>• nirs_channels</li><li>• time_channel</li><li>• event_channel</li><li>• sample_rate</li><li>• start_timestamp</li><li>• interval_times</li><li>• interval_span</li></ul>

## Details

Typically will only be called internally, but can be used to inject *mnirs* metadata into any data frame.

## Value

A [tibble](#) of class "mnirs". Metadata are stored as attributes and can be accessed with `attributes(data)`.

## Examples

```
data <- data.frame(
  A = 1:3,
  B = seq(10, 30, 10),
  C = seq(11, 33, 11)
)

attributes(data)

## inject metadata
nirs_data <- create_mnirs_data(
  data,
  nirs_channels = c("B", "C"),
  time_channel = "A",
  sample_rate = 1
)

attributes(nirs_data)
```

---

example_mnirs	<i>Get path to mnirs example files</i>
---------------	--

---

**Description**

Get path to *mnirs* example files

**Usage**

```
example_mnirs(file = NULL)
```

**Arguments**

file	Name of file as character string. If NULL, returns a vector of all available file names.
------	--

**Value**

A file path character string for selected example files stored in this package.

**Examples**

```
## lists all files
example_mnirs()

## partial matching will error if matches multiple
try(example_mnirs("moxy"))

example_mnirs("moxy_ramp")
```

---

extract_intervals	<i>Extract intervals from mnirs data</i>
-------------------	--

---

**Description**

Extract intervals from "*mnirs*" time series data, specifying interval start and end boundaries by time value, event label, lap number, or sample index.

**Usage**

```
extract_intervals(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  event_channel = NULL,
  sample_rate = NULL,
  start = NULL,
  end = NULL,
  span = list(c(-60, 60)),
  event_groups = c("distinct", "ensemble"),
  zero_time = FALSE,
  verbose = TRUE
)
```

**Arguments**

<code>data</code>	A data frame of class <i>"mnirs"</i> containing time series data and metadata.
<code>nirs_channels</code>	A character vector or a <code>list()</code> of character vectors of mNIRS channel names to operate on within each interval (see <i>Details</i> ). Names must match column names in data exactly. <ul style="list-style-type: none"> <li>• Must only be specified when <code>event_groups</code> contains <i>"ensemble"</i>- averaged intervals. If <code>event_groups = "distinct"</code> no channel processing occurs.</li> <li>• If <code>NULL</code> (default), channels are retrieved from <i>"mnirs"</i> metadata.</li> </ul>
<code>time_channel</code>	A character string giving the name of the time or sample column. Must match a column name in data exactly. <ul style="list-style-type: none"> <li>• If <code>NULL</code> (default), the <code>time_channel</code> metadata attribute of data is used.</li> </ul>
<code>event_channel</code>	An <i>optional</i> character string giving the name of an event/lap column. The column may contain character event labels or integer lap numbers. <ul style="list-style-type: none"> <li>• Required when using <code>by_label()</code> or <code>by_lap()</code> for start or end.</li> <li>• Retrieved from metadata if not defined explicitly.</li> </ul>
<code>sample_rate</code>	An <i>optional</i> numeric sample rate (Hz) used to bin time values for ensemble-averaging. If <code>NULL</code> , will be estimated from <code>time_channel</code> (see <i>Details</i> ).
<code>start</code>	Specifies where intervals begin. Either raw values — numeric for time values, character for event labels, explicit integer (e.g. 2L) for lap numbers — or created with <code>by_time()</code> , <code>by_label()</code> , <code>by_lap()</code> , or <code>by_sample()</code> .
<code>end</code>	Specifies where intervals end. Either raw values — numeric for time values, character for event labels, explicit integer (e.g. 2L) for lap numbers — or created with <code>by_time()</code> , <code>by_label()</code> , <code>by_lap()</code> , or <code>by_sample()</code> .
<code>span</code>	A one- or two-element numeric vector <code>c(before, after)</code> in units of <code>time_channel</code> , or a <code>list()</code> of such vectors. Applied additively to interval boundaries: <ul style="list-style-type: none"> <li>• When both <code>start</code> and <code>end</code> are specified: <code>span[1]</code> shifts start times, <code>span[2]</code> shifts end times.</li> <li>• When only <code>start</code> or only <code>end</code> is specified: both <code>span[1]</code> and <code>span[2]</code> apply as a window around the event).</li> </ul>

- A single *positive* value is recycled to shift the end times (e.g. `span = 60` -> `c(0, 60)`).
- A single *negative* value is recycled to shift the start times (e.g. `span = -60` -> `c(-60, 0)`).

event_groups	Either a character string or a <code>list()</code> of integer vectors specifying how to group intervals (see <i>Details</i> ).  "distinct" The default. Extract each interval as an independent data frame. "ensemble" Ensemble-average each specified <code>nirs_channel</code> across all detected intervals, returning a single data frame.  <code>list(c(1, 2), c(3, 4))</code> Ensemble-average each specified <code>nirs_channel</code> within each group and return one data frame per group.
zero_time	Logical. Default is FALSE. If TRUE, re-calculates numeric <code>time_channel</code> values to start from zero within each interval data frame.
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via <code>options(mnirs.verbose = FALSE)</code> .

## Details

### Interval specification:

Interval boundaries are specified using helper functions, or by passing raw values directly:

`by_time()` or **numeric** Time values in units of `time_channel`.

`by_label()` or **character** Strings to match in `event_channel`. All matching occurrences are returned.

`by_lap()` or **explicit integer (e.g. 2L)** Lap numbers to match in `event_channel`. Resolves to the first sample of each lap for start, and the last sample for end, or all samples of the lap if only one of either start or end is specified.

`by_sample()` Integer sample indices (row numbers).

Raw values supplied to start/end are auto-coerced:

- Numeric → `by_time()`
- Character → `by_label()`,
- Explicit integer (e.g. 2L) → `by_lap()`.
- Use `by_sample()` explicitly for sample indices.

start and end can use different specification types (e.g., start by label, end by time). When lengths differ, the shorter is recycled.

### The span window:

span applies an additive time shift to interval boundaries. A single numeric value is recycled: `span = 60` becomes `c(0, 60)` and `span = -60` becomes `c(-60, 0)`.

- start + end: `span[1]` shifts starts, `span[2]` shifts ends. For example, `start = by_time(30)`, `end = by_time(60)`, gives an interval of `[25, 70]`.
- start **only** or end **only**: both span values apply to the single boundary, like a window around an event. For example, `start = by_time(30)`, `span = c(-5, 60)` gives `[25, 90]`.

**Per-interval nirs\_channels for ensemble-averaging:**

When `event_groups = "ensemble"` or a list of numeric grouped intervals, `nirs_channels` can be specified as a list of column names to override ensemble-averaging across interval. For example, to exclude a bad channel in one interval:

```
nirs_channels = list(
  c(A, B, C),
  c(A, C) ## channel "B" is excluded
)
```

If all grouped intervals can include all `nirs_channels`, or if `event_groups = "distinct"`, a single `nirs_channels` character vector can be supplied and recycled to all groups, or left as `NULL` for channels to be taken from `"mnirs"` metadata.

**Grouping intervals:**

`event_groups` controls whether extracted intervals are returned as distinct data frames or ensemble-averaged.

"distinct" The default. Extract each interval and return a list of independent data frames.

"ensemble" Ensemble-average each specified `nirs_channel` across all detected intervals and return a one-item list with a single data frame.

`list(c(1, 2), c(3, 4))` Ensemble-average each specified `nirs_channel` within each group and return a list with one data frame for each group. Any intervals detected but not specified in `event_groups` are returned as distinct.

`event_groups` lists can be named (e.g. `list(low = c(1, 2), high = c(3, 4))`) and will pass those names to the returned list of data frames.

When `event_groups` is a list of numeric interval numbers, list items in `nirs_channels` and `span` are recycled to the number of groups. If lists are only partially specified, the final item is recycled forward as needed. Extra items are ignored.

**Value**

A named `list()` of `tibbles` of class `"mnirs"`, with metadata available via `attributes()`.

**Examples**

```
## read example data
data <- read_mnirs(
  example_mnirs("train.red"),
  nirs_channels = c(
    smo2_left = "SmO2 unfiltered",
    smo2_right = "SmO2 unfiltered"
  ),
  time_channel = c(time = "Timestamp (seconds passed)"),
  zero_time = TRUE,
  verbose = FALSE
) |>
  resample_mnirs(verbose = FALSE) ## avoid issues ensemble-averaging irregular samples

## ensemble-average across multiple intervals
interval_list <- extract_intervals(
```

```

    data,                                ## channels recycled to all intervals by default
    nirs_channels = c(smo2_left, smo2_right),
    start = by_time(368, 1093), ## manually identified interval start times
    span = c(-20, 90),           ## include the last 180-sec of each interval (recycled)
    event_groups = "ensemble",   ## ensemble-average across two intervals
    zero_time = TRUE             ## re-calculate common time to start from `0`
  )

interval_list[[1L]]

if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(interval_list[[1L]], time_labels = TRUE) +
    ggplot2::geom_vline(xintercept = 0, linetype = "dotted")
}

```

---

 filter\_butter

 Apply a Butterworth digital filter
 

---

## Description

Apply a Butterworth digital filter to vector data with `signal::butter()` and `signal::filtfilt()` which handles 'edges' better at the start and end of the data.

## Usage

```

filter_butter(
  x,
  order = 2L,
  W,
  type = c("low", "high", "stop", "pass"),
  edges = c("rev", "rep1", "none"),
  na.rm = FALSE,
  ...
)

```

## Arguments

x	A numeric vector.
order	An integer defining the filter order ( <i>default</i> order = 2).
W	A one- or two-element numeric vector defining the filter cutoff frequency(ies) as a fraction of the Nyquist frequency (see <i>Details</i> ).
type	A character string indicating the digital filter type (see <i>Details</i> ). "low" For a <i>low-pass</i> filter (the <i>default</i> ). "high" For a <i>high-pass</i> filter.

	"stop" For a <i>stop-band</i> (band-reject) filter.
	"pass" For a <i>pass-band</i> filter.
edges	A character string indicating edge detection padding for x.
	"rev" Will pad x with the preceding 5% data in reverse sequence ( <i>the default</i> ).
	"rep1" Will pad x by repeating the last preceding value.
	"none" Will return the unpadded <code>signal::filtfilt()</code> output.
na.rm	Logical; default is FALSE, propagates any NAs to the returned vector. If TRUE, ignores NAs and processes available valid samples within the local window. May return errors or warnings. (see <i>Details</i> ).
...	Additional method-specific arguments must be specified (see <i>Details</i> ).

### Details

Applies a centred (two-pass symmetrical) Butterworth digital filter from `signal::butter()` and `signal::filtfilt()`.

Filter type defines how the desired signal frequencies are either passed or rejected from the output signal. *Low-pass* and *high-pass* filters allow only frequencies *lower* or *higher* than the cutoff frequency  $W$  to be passed through as the output signal, respectively. *Stop-band* defines a critical range of frequencies which are rejected from the output signal. *Pass-band* defines a critical range of frequencies which are passed through as the output signal.

The filter order (number of passes) is defined by `order`, typically in the range `order = [1, 10]`. Higher filter order tends to capture more rapid changes in amplitude, but also causes more distortion around those change points in the signal. General advice is to use the lowest filter order which sufficiently captures the desired rapid responses in the data.

The critical (cutoff) frequency is defined by  $W$ , a numeric value for *low-pass* and *high-pass* filters, or a two-element vector `c(low, high)` defining the lower and upper bands for *stop-band* and *pass-band* filters.  $W$  represents the desired fractional cutoff frequency in the range  $W = [0, 1]$ , where 1 is the Nyquist frequency, i.e., half the sample rate of the data in Hz.

Missing values (NA) in `x` will cause an error unless `na.rm = TRUE`. Then NAs will be ignored and passed through to the returned vector.

### Value

A numeric vector the same length as `x`.

### See Also

`signal::filtfilt()`, `signal::butter()`

### Examples

```
set.seed(13)
sin <- sin(2 * pi * 1:150 / 50) * 20 + 40
noise <- rnorm(150, mean = 0, sd = 6)
noisy_sin <- sin + noise
without_edge_detection <- filter_butter(
  x = noisy_sin,
```

```

    order = 2,
    W = 0.1,
    edges = "none"
  )
  with_edge_detection <- filter_butter(
    x = noisy_sin,
    order = 2,
    W = 0.1,
    edges = "rep1"
  )

ggplot2::ggplot(data.frame(), ggplot2::aes(x = seq_along(noise))) +
  theme_mnirs() +
  scale_colour_mnirs(name = NULL) +
  ggplot2::geom_line(ggplot2::aes(y = noisy_sin)) +
  ggplot2::geom_line(
    ggplot2::aes(y = without_edge_detection, colour = "without_edge_detection")
  ) +
  ggplot2::geom_line(
    ggplot2::aes(y = with_edge_detection, colour = "with_edge_detection")
  )

```

---

 filter\_ma

*Apply a moving average filter*


---

### Description

Apply a simple moving average smoothing filter to vector data. `filter_moving_average()` is an alias of `filter_ma()`.

### Usage

```

filter_ma(
  x,
  t = seq_along(x),
  width = NULL,
  span = NULL,
  partial = FALSE,
  na.rm = FALSE,
  verbose = TRUE,
  ...
)

filter_moving_average(
  x,
  t = seq_along(x),
  width = NULL,

```

```

span = NULL,
partial = FALSE,
na.rm = FALSE,
verbose = TRUE,
...
)

```

### Arguments

x	A numeric vector of the response variable.
t	An <i>optional</i> numeric vector of the predictor variable (time or sample number). Default is seq_along(x).
width	An integer defining the local window in number of samples centred on idx, between [idx - floor(width/2), idx + floor(width/2)].
span	A numeric value defining the local window timespan around idx in units of time_channel or t, between [t - span/2, t + span/2].
partial	Logical; default is FALSE, requires local windows to have complete number of samples specified by width or span. If TRUE, processes available samples within the local window. See <i>Details</i> .
na.rm	Logical; default is FALSE, propagates any NAs to the returned vector. If TRUE, ignores NAs and processes available valid samples within the local window. May return errors or warnings. (see <i>Details</i> ).
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via options(mnirs.verbose = FALSE).
...	Additional arguments.

### Details

#### Rolling window:

Applies a centred (symmetrical) moving average filter in a local window, defined by either width as the number of samples around idx between [idx - floor(width/2), idx + floor(width/2)]. Or by span as the timespan in units of time\_channel between [t - span/2, t + span/2].

#### Partial windows:

The default partial = FALSE requires a complete number of samples specified by width or span (estimated from the sample rate of t when span is used). NA is returned if fewer samples are present in the local window.

Setting partial = TRUE allows computation with only a single valid sample, such as at edge conditions. But these values will be more sensitive to noise and should be used with caution.

#### Missing values:

na.rm controls whether missing values (NAs) within each local window are either propagated to the returned vector when na.rm = FALSE (the default), or ignored before processing if na.rm = TRUE.

**Value**

A numeric vector the same length as x.

**Examples**

```
x <- c(1, 3, 2, 5, 4, 6, 5, 7)
t <- c(0, 1, 2, 4, 5, 6, 7, 10) ## irregular time with gaps

## width: centred window of 3 samples
filter_ma(x, width = 3)

## partial = TRUE fills edge values with a narrower window
filter_ma(x, width = 3, partial = TRUE)

## span: centred window of 2 time-units (accounts for irregular sampling)
filter_ma(x, t, span = 2)

## na.rm = FALSE (default): any NA in the window propagates to the result
x_na <- c(1, NA, 3, 4, 5, NA, 7, 8)
filter_ma(x_na, width = 3, na.rm = FALSE)

## na.rm = TRUE: skip NAs and return the local mean of local valid values
filter_ma(x_na, width = 3, partial = TRUE, na.rm = TRUE)
```

---

 filter\_mnirs

*Filter a data frame*


---

**Description**

Apply digital filtering/smoothing to numeric vector data within a data frame using either:

1. A cubic smoothing spline.
2. A Butterworth digital filter.
3. A simple moving average.

**Usage**

```
filter_mnirs(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  method = c("smooth_spline", "butterworth", "moving_average"),
  na.rm = FALSE,
  verbose = TRUE,
  ...
)
```

**Arguments**

data	A data frame of class "mnirs" containing time series data and metadata.
nirs_channels	A character vector giving the names of mNIRS columns to operate on. Must match column names in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the nirs_channels metadata attribute of data is used.</li> </ul>
time_channel	A character string giving the name of the time or sample column. Must match a column name in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the time_channel metadata attribute of data is used.</li> </ul>
method	A character string indicating how to filter the data (see <i>Details</i> ). "smooth_spline" Fits a cubic smoothing spline. "butterworth" Uses a centred Butterworth digital filter. "moving_average" Uses a centred moving average filter.
na.rm	Logical; default is FALSE, propagates any NAs to the returned vector. If TRUE, ignores NAs and processes available valid samples within the local window. May return errors or warnings. (see <i>Details</i> ).
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via options(mnirs.verbose = FALSE).
...	Additional method-specific arguments must be specified (see <i>Details</i> ).

**Details****filtering method:**

method = "smooth\_spline":

Applies a non-parametric cubic smoothing spline from `stats::smooth.spline()`. Smoothing is defined by the parameter `spar`, which can be left as NULL and automatically determined via penalised log likelihood. This usually works well for responses occurring on the order of minutes or longer. `spar` can be specified typically, but not necessarily, in the range `spar = [0, 1]`. Additional arguments (...) accepted when `method = "smooth_spline"`:

`spar` A numeric smoothing parameter passed to `stats::smooth.spline()`. If NULL (*default*), automatically determined via penalised log likelihood.

method = "butterworth":

Applies a centred (two-pass symmetrical) Butterworth digital filter from `signal::butter()` and `signal::filtfilt()`.

Filter type defines how the desired signal frequencies are either passed or rejected from the output signal. *Low-pass* and *high-pass* filters allow only frequencies *lower* or *higher* than the cutoff frequency, respectively to be passed through to the output signal. *Stop-band* defines a critical range of frequencies which are rejected from the output signal. *Pass-band* defines a critical range of frequencies which are passed through as the output signal.

The filter order (number of passes) is defined by `order`, typically in the range `order = [1, 10]`. Higher filter order tends to capture more rapid changes in amplitude, but also causes more distortion around those change points in the signal. General advice is to use the lowest filter order which sufficiently captures the desired rapid responses in the data.

The critical (cutoff) frequency can be defined by  $W$ , a numeric value for *low-pass* and *high-pass* filters, or a two-element vector  $c(\text{low}, \text{high})$  defining the lower and upper bands for *stop-band* and *pass-band* filters.  $W$  represents the desired fractional cutoff frequency in the range  $W = [0, 1]$ , where 1 is the Nyquist frequency, i.e., half the `sample_rate` of the data in Hz.

Alternatively, the cutoff frequency can be defined by `fc` and `sample_rate` together. `fc` represents the desired cutoff frequency directly in Hz, and `sample_rate` is the sample rate of the recorded data in Hz. Where  $W = fc / (\text{sample\_rate} / 2)$ .

Only one of either  $W$  or `fc` should be defined. If both are defined,  $W$  will be preferred over `fc`.

Additional arguments (...) accepted when `method = "butterworth"`:

`order` An integer for the filter order (*default* 2).

`W` A numeric fractional cutoff frequency within  $[0, 1]$ . One of either  $W$  or `fc` must be specified.

`fc` A numeric absolute cutoff frequency in Hz. Used with `sample_rate` to compute  $W$ .

`sample_rate` A numeric sample rate in Hz. Will be taken from metadata or estimated from `time_channel` if not defined.

`type` A character string specifying filter type, one of:  $c(\text{"low"}, \text{"high"}, \text{"stop"}, \text{"pass"})$  ("low" is the default).

`edges` A character string specifying the edge padding, one of:  $c(\text{"rev"}, \text{"rep1"}, \text{"none"})$  ("rev" is the default). See [filter\\_butter\(\)](#).

`method = "moving_average"`:

Applies a centred (symmetrical) moving average filter in a local window, defined by either `width` as the number of samples around `idx` between  $[\text{idx} - \text{floor}(\text{width}/2), \text{idx} + \text{floor}(\text{width}/2)]$ . Or by `span` as the timespan in units of `time_channel` between  $[\text{t} - \text{span}/2, \text{t} + \text{span}/2]$ .

Additional arguments (...) accepted when `method = "moving_average"`:

`width` or `span` Either an integer number of samples, or a numeric time duration in units of `time_channel` within the local window. One of either `width` or `span` must be specified.

`partial` Logical; FALSE by default, only returns values where a full window of valid (non-NA) samples are available. If TRUE, ignores NA and allows calculation over partial windows at the edges of the data.

### Missing values:

Missing values (NA) in `nirs_channels` will cause an error for `method = "smooth_spline"` or `"butterworth"`, unless `na.rm = TRUE`. Then NAs will be ignored and passed through to the returned data.

For `method = "moving_average"`, `na.rm` controls whether NAs within each local window are either propagated to the returned vector when `na.rm = FALSE` (the default), or ignored before processing if `na.rm = TRUE`.

### Value

A [tibble](#) of class `"mnirs"` with metadata available with `attributes()`.

### Examples

```
## read example data and clean for outliers
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2 = "SmO2 Live"),
  time_channel = c(time = "hh:mm:ss"),
```

```

    verbose = FALSE
  ) |>
  replace_mnirs(
    invalid_values = c(0, 100),
    outlier_cutoff = 3,
    width = 10,
    verbose = FALSE
  )

data

data_filtered <- filter_mnirs(
  data,                                ## blank channels will be retrieved from metadata
  method = "butterworth", ## Butterworth digital filter is a common choice
  order = 2,                        ## filter order number
  W = 0.02,                          ## filter fractional critical frequency `[0, 1]`
  type = "low",                      ## specify a "low-pass" filter
  na.rm = TRUE                       ## explicitly ignore NAs
)

## note the smoothed `smo2` values
data_filtered

if (requireNamespace("ggplot2", quietly = TRUE)) {
  ## plot filtered data and add the raw data back to the plot to compare
  plot(data_filtered, time_labels = TRUE) +
    ggplot2::geom_line(
      data = data,
      ggplot2::aes(y = smo2, colour = "smo2"), alpha = 0.4
    )
}

```

---

format\_hmmss

*Format timespan data as h:mm:ss*


---

### Description

Convert numeric timespan data to h:mm:ss format for pretty plotting. Inspired by `ggplot2::scale_x_time()`.

### Usage

```
format_hmmss(x)
```

### Arguments

x                    A numeric vector.

**Details**

If all values are less than 3600 (1 hour), then format is returned as mm:ss. If any value is greater than 3600, format is returned as h:mm:ss with leading zeroes.

**Value**

A character vector the same length as x.

**Examples**

```
x <- 0:120
y <- sin(2 * pi * x / 15) + rnorm(length(x), 0, 0.2)

ggplot2::ggplot(data.frame(x, y), ggplot2::aes(x, y)) +
  theme_mnirs() +
  ggplot2::scale_x_continuous(
    breaks = breaks_timespan(),
    labels = format_hmmss
  ) +
  ggplot2::geom_line()
```

---

moxy\_intervals.csv      *0.5 Hz Moxy onboard export*

---

**Description**

Exported from Moxy onboard recording at 0.5 Hz no smoothing. Containing four 4-minute cycling work intervals, placed on the vastus lateralis muscle site.

**Format**

.csv file with seven columns and 936 rows:

**mm-dd** Recording date (dd-MMM format).

**hh:mm:ss** Recording time of day (hh:mm:ss format).

**SmO2 Live** Muscle oxygen saturation, raw signal (%).

**SmO2 Averaged** Muscle oxygen saturation, rolling average (%).

**THb** Total haemoglobin (arbitrary units).

**Lap** Lap marker (integer). Not typically in use.

**Session Ct** Session count of recordings.

Channel mapping for `read_mnirs()`:

- `nirs_channels = c("SmO2 Live", "SmO2 Averaged", "THb")`
- `time_channel = c("hh:mm:ss")`
- `interval_times = list(start = c(124, 486, 848, 1210), end = c(364, 726, 1088, 1450))`

**Source**

Moxy Monitor (Fortiori Design LLC), exported via Moxy Portal App. (<https://www.moxymonitor.com/>)

**See Also**

[read\\_mnirs\(\)](#), [example\\_mnirs\(\)](#)

**Examples**

```
example_mnirs("moxy_intervals")
```

---

moxy\_ramp.xlsx

*2 Hz PerfPro export of Moxy data*

---

**Description**

Exported from PerfPro Studio software, recorded at 0.5 Hz no smoothing and exported at 2 Hz. Containing a ramp incremental cycling protocol, placed on bilateral vastus lateralis muscle sites. Intentional data errors (outliers, invalid values, and missing NA values) have been introduced to demonstrate *mnirs* cleaning functions.

**Format**

.xlsx file with five columns and 2202 rows:

**mm-dd** Recording date (dd-MMM format).

**hh:mm:ss** Time of day (hh:mm:ss format).

**SmO2 Live** Muscle oxygen saturation, left leg (%). Contains simulated erroneous and missing samples.

**SmO2 Live(2)** Muscle oxygen saturation, right leg (%).

**Lap** Lap marker (integer).

Channel mapping for [read\\_mnirs\(\)](#):

- `nirs_channels = c("SmO2 Live", "SmO2 Live(2)")`
- `time_channel = c("hh:mm:ss")`
- `event_channel = c("Lap")`
- `interval_times = list(start = c(204, 878))` (occlusion and hyperaemia)

**Source**

Moxy Monitor (Fortiori Design LLC), exported via PerfPro Studio desktop software (<https://perfprostudio.com/>).

**See Also**

[read\\_mnirs\(\)](#), [example\\_mnirs\(\)](#)

**Examples**

```
example_mnirs("moxy_ramp")
```

---

palette_mnirs	<i>Custom mnirs colour palette</i>
---------------	------------------------------------

---

**Description**

Custom *mnirs* colour palette

**Usage**

```
palette_mnirs(...)
```

**Arguments**

... Either a single numeric specifying the number of colours to return, or character strings specifying colour names. If empty, all colours are returned.

**Value**

Named (when selecting by name) or unnamed character vector of hex colours.

**See Also**

[theme\\_mnirs\(\)](#), [scale\\_colour\\_mnirs\(\)](#)

**Examples**

```
scales::show_col(palette_mnirs())  
scales::show_col(palette_mnirs(2))  
scales::show_col(palette_mnirs("red", "orange"))
```

plot.mnirs

*Plot mnirs objects***Description**

Create a simple plot for objects returned from `create_mnirs_data()`.

**Usage**

```
## S3 method for class 'mnirs'
plot(
  x,
  points = FALSE,
  time_labels = FALSE,
  n.breaks = 5,
  na.omit = FALSE,
  ...
)
```

**Arguments**

<code>x</code>	Object of class <i>"mnirs"</i> returned from <code>create_mnirs_data()</code>
<code>points</code>	Logical. Default is FALSE. If TRUE displays <code>ggplot2::geom_points()</code> . Otherwise only <code>ggplot2::geom_lines()</code> is displayed.
<code>time_labels</code>	Logical. Default is FALSE. If TRUE displays x-axis time values formatted as <i>"hh:mm:ss"</i> using <code>format_hmmss()</code> . Otherwise, x-axis values are displayed as numeric.
<code>n.breaks</code>	A numeric value specifying the number of breaks in both x- and y-axes. Default is 5.
<code>na.omit</code>	Logical. Default is FALSE. If TRUE omits missing (NA) and non-finite c(Inf, -Inf, NaN) from display.
<code>...</code>	Additional arguments.

**Value**

A `ggplot2` object.

**Examples**

```
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2_left = "SmO2 Live",
                   smo2_right = "SmO2 Live(2)"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
)
```

```
## note the options to display time values as `h:mm:ss` with 8 breaks  
plot(data, time_labels = TRUE, n.breaks = 8)
```

---

portamon-oxcap.xlsx    *10 Hz Artinis Oxysoft export recorded with Portamon*

---

## Description

Exported from Artinis Oxysoft, recorded on Portamon at 10 Hz on the vastus lateralis muscle. Containing two trials of repeated occlusion oxidative capacity testing, each with 17 occlusions.

## Format

.xlsx file with header metadata and six columns and 7943 rows:

- Column 1** Sample index (divide by sample rate for seconds).
- Column 2** tHb: total haemoglobin concentration change ( $\mu\text{M}$ ).
- Column 2** O2Hb: oxyhaemoglobin concentration change ( $\mu\text{M}$ ).
- Column 3** HHb: deoxyhaemoglobin concentration change ( $\mu\text{M}$ ).
- Column 4** Event marker (character).
- Column 5** Unmarked event label (character).

Channel mapping for `read_mnirs()`:

- `nirs_channels = c(THb = 2, HHb = 3, O2Hb = 4)`
- `time_channel = c(sample = 1)`
- `event_channel = c(event = 5, label = "col_6")`

## Source

Artinis Medical Systems. Portamon, exported via Oxysoft desktop software (<https://www.artinis.com/>)

## See Also

`read_mnirs()`, `example_mnirs()`

## Examples

```
example_mnirs("portamon")
```

---

 read\_mnirs

*Read mnirs data from file*


---

### Description

Import time-series data exported from common muscle NIRS (mNIRS) devices and return a tibble of class "mnirs" with the selected signal channels and metadata.

### Usage

```
read_mnirs(
  file_path,
  nirs_channels = NULL,
  time_channel = NULL,
  event_channel = NULL,
  sample_rate = NULL,
  add_timestamp = FALSE,
  zero_time = FALSE,
  keep_all = FALSE,
  verbose = TRUE
)
```

### Arguments

file_path	Path of the data file to import. Supported file extensions are ".xlsx", ".xls", and ".csv".
nirs_channels	A character vector of one or more column names containing mNIRS signals to import. Names must match the file header exactly. <ul style="list-style-type: none"> <li>• If NULL (default), read_mnirs() attempts to detect the device from the file contents and use a known nirs_channel name.</li> <li>• A <i>named</i> character vector can be used to rename columns on import, in the form c(renamed = "original_name").</li> </ul>
time_channel	A character string giving the name of the time (or sample) column to import. The name must match the file header exactly. <ul style="list-style-type: none"> <li>• If NULL (default), read_mnirs() attempts to identify a time-like column automatically (by known device defaults and/or time-formatted values).</li> <li>• A <i>named</i> character vector can be used to rename the column on import, in the form c(time = "original_name").</li> </ul>
event_channel	An <i>optional</i> character string giving the name of an event/lap column to import. Names must match the file header exactly. A named character vector can be used to rename the column on import in the form c(event = "original_name").
sample_rate	An <i>optional</i> numeric sample rate in Hz. If left blank (NULL), the sample rate is estimated from time_channel (see <i>Details</i> ).

add_timestamp	A logical. Default is FALSE. If TRUE and if the source data contain an absolute date-time (POSIXct) time value, will add a "timestamp" column in addition to the specified time_channel as a numeric time column.
zero_time	Logical. Default is FALSE. If TRUE, re-calculates numeric time_channel values to start from zero.
keep_all	Logical. Default is FALSE. Will keep only the channels explicitly specified in nirs_channels, time_channel, and event_channel. If TRUE will keep all columns found in the file data table. <ul style="list-style-type: none"> <li>• If no nirs_channels are specified and the file format is recognised, all columns in the file data table will be returned, as an exploratory option.</li> </ul>
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via options(mnirs.verbose = FALSE).

## Details

### Header detection:

read\_mnirs() searches the file for a header row containing the requested channel names. The header row does not need to be the first row in the file.

- If duplicate column names exist, columns are matched in the order they appear and renamed with unique strings.
- Columns without a header name in the source file will be renamed to col\_\*, where \* is the numeric column number in which they appear in the file (e.g. col\_6). This applies to *Artinis Oxysoft* event label columns, which do not have a column header and must be identified manually.

### Renaming channels:

A named character vector can be specified to rename nirs\_channels, time\_channel, and event\_channel, in the form c(renamed = "original\_name"). The "original\_name" must match the contents of the file data table header row exactly.

### Time parsing:

time\_channel will be converted to numeric for analysis.

- If time\_channel is a date-time (POSIXct) format, it will be converted to numeric and re-based to start from 0, regardless of zero\_time.
- Some devices export a sample index rather than time values. In those cases, if an export sample\_rate is detected in the file metadata (e.g. *Artinis Oxysoft* exports), read\_mnirs() will create or overwrite a "time" column in seconds derived from the sample index and the detected sample\_rate.

### Sample rate:

If sample\_rate is not specified, it is estimated from differences in time\_channel. If time\_channel is actually a sample index, as described above, this may erroneously be estimated at 1 Hz. sample\_rate should be specified explicitly in this case.

### Data cleaning:

Entirely empty rows and columns are removed. Invalid values (e.g. c(NaN, Inf)) are standardized to NA. A warning will be displayed when irregular sampling is detected (e.g. non-monotonic, repeated, or unequal time values), if verbose = TRUE.

**Value**

A **tibble** of class "mnirs". Metadata are stored as attributes and can be accessed with `attributes(data)`.

**Examples**

```
read_mnirs(
  file_path = example_mnirs("moxy_ramp"), ## call an example data file
  nirs_channels = c(
    smo2_left = "SmO2 Live",           ## identify and rename channels
    smo2_right = "SmO2 Live(2)"
  ),
  time_channel = c(time = "hh:mm:ss"), ## date-time format will be converted to numeric
  event_channel = NULL,                ## leave blank if unused
  sample_rate = NULL,                  ## if blank, will be estimated from time_channel
  add_timestamp = FALSE,               ## omit a date-time timestamp column
  zero_time = TRUE,                    ## recalculate time values from zero
  keep_all = FALSE,                   ## return only the specified data channels
  verbose = TRUE                       ## show warnings & messages
)
```

---

 replace\_mnirs

*Replace outliers, invalid, and missing values in mnirs data*


---

**Description**

Detect and replace local outliers, specified invalid values, and missing NA values across `nirs_channels` within an "mnirs" data frame. `replace_mnirs()` operates on a data frame, extending the vectorised functions:.

`replace_invalid()` detects specified invalid values or range cutoffs in a numeric vector and replace them with the local median value or NA.

`replace_outliers()` detects local outliers in a numeric vector using a Hampel filter and replaces with the local median value or NA.

`replace_missing()` detects missing (NA) values in a numeric vector and replaces via interpolation.

**Usage**

```
replace_mnirs(
  data,
  nirs_channels = NULL,
  time_channel = NULL,
  invalid_values = NULL,
  invalid_above = NULL,
  invalid_below = NULL,
  outlier_cutoff = NULL,
  width = NULL,
  span = NULL,
```

```

    method = c("linear", "median", "locf", "none"),
    verbose = TRUE
  )

  replace_invalid(
    x,
    t = seq_along(x),
    invalid_values = NULL,
    invalid_above = NULL,
    invalid_below = NULL,
    width = NULL,
    span = NULL,
    method = c("median", "none"),
    bypass_checks = FALSE,
    verbose = TRUE
  )

  replace_outliers(
    x,
    t = seq_along(x),
    outlier_cutoff = 3,
    width = NULL,
    span = NULL,
    method = c("median", "none"),
    bypass_checks = FALSE,
    verbose = TRUE
  )

  replace_missing(
    x,
    t = seq_along(x),
    width = NULL,
    span = NULL,
    method = c("linear", "median", "locf"),
    bypass_checks = FALSE,
    verbose = TRUE,
    ...
  )

```

### Arguments

- |               |  |
|---------------|--|
| data          | A data frame of class <i>"mnirs"</i> containing time series data and metadata.   |
| nirs_channels | A character vector giving the names of mNIRS columns to operate on. Must match column names in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the <code>nirs_channels</code> metadata attribute of data is used.</li> </ul> |
| time_channel  | A character string giving the name of the time or sample column. Must match a column name in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the <code>time_channel</code> metadata attribute of data is used.</li> </ul>    |

invalid_values	A numeric vector of invalid values to be replaced, e.g. <code>invalid_values = c(0, 100, 102.3)</code> . Default NULL will not replace invalid values.
invalid_above, invalid_below	Numeric values each specifying cutoff values, above or below which (respectively) will be replaced, <i>inclusive</i> of the specified cutoff values.
outlier_cutoff	A numeric value for the local outlier threshold, as the number of standard deviations from the local median. <ul style="list-style-type: none"> <li>• Default NULL will not replace outliers.</li> <li>• Lower values are more sensitive and flag more outliers; higher values are more conservative.</li> <li>• <code>outlier_cutoff = 3</code> Pearson's 3 sigma edit rule. <code>outlier_cutoff = 2</code> approximates a Tukey-style 1.5×IQR rule. <code>outlier_cutoff = 0</code> Tukey's median filter.</li> </ul>
width	An integer defining the local window in number of samples centred on <code>idx</code> , between <code>[idx - floor(width/2), idx + floor(width/2)]</code> .
span	A numeric value defining the local window timespan around <code>idx</code> in units of <code>time_channel</code> or <code>t</code> , between <code>[t - span/2, t + span/2]</code> .
method	A character string indicating how to handle NA replacement (see <i>Details</i> ): <p>"linear" Replaces NAs via linear interpolation (the <i>default</i>) using <code>stats::approx()</code>.</p> <p>"median" Replaces NAs with the local median of valid values within a centred window defined by <code>width</code> or <code>span</code>.</p> <p>"locf" "<i>Last observation carried forward</i>". Replaces NAs with the most recent valid value to the left for trailing NAs or to the right for leading NAs, using <code>stats::approx()</code>.</p> <p>"none" Returns NAs without replacement.</p>
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via <code>options(mnirs.verbose = FALSE)</code> .
x	A numeric vector of the response variable.
t	An <i>optional</i> numeric vector of the predictor variable (time or sample number). Default is <code>seq_along(x)</code> .
bypass_checks	Logical allowing wrapper functions to bypass redundant checks and validations.
...	Additional arguments.

## Details

### Automatic channel detection:

`nirs_channels` and `time_channel` are retrieved automatically from "*mnirs*" metadata if not specified explicitly. Columns in data not listed in `nirs_channels` are passed through unprocessed.

### The rolling window:

`replace_outliers()` and `replace_missing()` (when `method = "median"`) operate over a local rolling window for outlier detection and median interpolation. The window is specified by either `width` as the number of samples, or `span` as the timespan in units of `time_channel`. A partial window is calculated at the edges of the data.

**Replace invalid values with** `replace_invalid()`:

Specific `invalid_values` can be replaced, such as `c(0, 100, 102.3)`. Data ranges can be replaced with cutoff values specified by `invalid_above` and `invalid_below`, where any values higher or lower than the specified cutoff values (respectively) will be replaced, *inclusive* of the cutoff values themselves.

**Outlier detection with** `replace_outliers()`:

Rolling local medians are computed across `x` within a window defined by `width` (number of samples) or `span` (timespan in units of `t`).

Outliers are detected with robust median absolute deviation (MAD), adapted from `pracma::hampel()`. Deviations equal to or less than the smallest absolute time series difference in `x` are excluded, to avoid flagging negligible differences where local data have minimal or zero variation.

**Replacement behaviour:**

Values of `x` outside the local bounds defined by `outlier_cutoff` are identified as outliers and either replaced with the local median (`method = "median"`, the *default*) or set to NA (`method = "none"`).

Existing NA values in `x` are *not* replaced. They are passed through to the returned vector. See `replace_missing()`.

**Choosing** `outlier_cutoff`:

`outlier_cutoff` is the number of (MAD-normalised) standard deviations from the local median. Higher values are more conservative; lower values flag more outliers.

- `outlier_cutoff = 3` — Pearson's 3 sigma edit rule (default).
- `outlier_cutoff = 2` — approximately Tukey-style 1.5×IQR rule.
- `outlier_cutoff = 0` — Tukey's median filter (every point replaced by local median).

**Interpolation with** `replace_missing()`:

`method = "linear"` and `method = "locf"` use `stats::approx()` with `rule = 2`, so leading NAs are filled by *"nocb"* (*"next observation carried backward"*) and trailing NAs by *"locf"*.

`method = "median"` calculates the local median of valid (non-NA) values to either side of NAs, within a window defined by `width` (number of samples) or `span` (timespan in units of `t`). Sequential NAs are all replaced by the same median value.

**Edge behaviour for** `method = "median"`:

If there are no valid values within `span` to one side of the NA, the median of the other side is used (i.e. for leading and trailing NAs). If there are no valid values within either side, the first valid sample on either side is used (equivalent to `replace_missing(x, width = 1)`).

**Value**

`replace_mnirs()` return a [tibble](#) of class "mnirs" with metadata available via `attributes()`.

`replace_invalid()` returns a numeric vector the same length as `x` with invalid values replaced.

`replace_outliers()` returns a numeric vector the same length as `x` with outliers replaced.

`replace_missing()` returns a numeric vector the same length as `x` with missing values replaced.

**Examples**

```

## vectorised operations
x <- c(1, 999, 3, 4, 999, 6)
replace_invalid(x, invalid_values = 999, width = 3, method = "median")

(x_na <- replace_outliers(x, outlier_cutoff = 3, width = 3, method = "none"))

replace_missing(x_na, method = "linear")

## read example data
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2 = "SmO2 Live"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
)

## clean data
data_clean <- replace_mnirs(
  data,                ## channels retrieved from metadata
  invalid_values = 0,  ## known invalid values in the data
  invalid_above = 90,  ## remove data spikes above 90
  outlier_cutoff = 3,  ## Pearson's 3 sigma edit rule
  width = 10,         ## window for outlier detection and interpolation
  method = "linear"   ## linear interpolation over NAs
)

if (requireNamespace("ggplot2", quietly = TRUE)) {
  ## plot original and show where values have been replaced
  ## ignore warning about replacing the existing colour scale
  plot(data, time_labels = TRUE) +
    ggplot2::scale_colour_manual(
      name = NULL,
      breaks = c("smo2", "replaced"),
      values = palette_mnirs(2)
    ) +
    ggplot2::geom_point(
      data = data[data_clean$smo2 != data$smo2, ],
      ggplot2::aes(y = smo2, colour = "replaced"),
      na.rm = TRUE
    ) +
    ggplot2::geom_line(
      data = {
        data_clean[!is.na(data$smo2), "smo2"] <- NA
        data_clean
      },
      ggplot2::aes(y = smo2, colour = "replaced"),
      linewidth = 1, na.rm = TRUE
    )
}

```

---

resample_mnirs	<i>Re-sample an mnirs data frame</i>
----------------	--------------------------------------

---

### Description

Up- or down-sample an *"mnirs"* data frame to a new sample rate, filling new samples via nearest-neighbour matching or interpolation.

### Usage

```
resample_mnirs(
  data,
  time_channel = NULL,
  sample_rate = NULL,
  resample_rate = sample_rate,
  method = c("locf", "linear", "none"),
  verbose = TRUE
)
```

### Arguments

<code>data</code>	A data frame of class <i>"mnirs"</i> containing time series data and metadata.
<code>time_channel</code>	A character string giving the name of the time or sample column. Must match a column name in <code>data</code> exactly. <ul style="list-style-type: none"> <li>If <code>NULL</code> (default), the <code>time_channel</code> metadata attribute of <code>data</code> is used.</li> </ul>
<code>sample_rate</code>	A numeric sample rate in Hz. <ul style="list-style-type: none"> <li>If <code>NULL</code> (default), the <code>sample_rate</code> metadata attribute of <code>data</code> will be used if detected, or the sample rate will be estimated from <code>time_channel</code>.</li> </ul>
<code>resample_rate</code>	An <i>optional</i> sample rate (Hz) for the output data frame. If <code>NULL</code> ( <i>default</i> ) re-samples to the existing <code>sample_rate</code> , which regularises any irregular samples without changing the rate.
<code>method</code>	A character string specifying how new samples are filled. Default is <i>"locf"</i> (see <i>Details</i> for more on each method): <ul style="list-style-type: none"> <li><i>"locf"</i> (<i>"Last observation carried forward"</i>). Fills new and missing samples with the most recent valid non-NA value to the left, or the nearest valid value to the right for leading NAs. Safe for numeric, integer, and character columns.</li> <li><i>"linear"</i> Fills new and missing samples via linear interpolation using <code>stats::approx()</code>. Suitable for numeric columns only; non-numeric columns will fall back to <i>"locf"</i> behaviour.</li> <li><i>"none"</i> Matches each new sample to the nearest original <code>time_channel</code> value within half a sample-interval tolerance, without any interpolation. New samples that fall between original values are returned as NA.</li> </ul>

`verbose` Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via `options(mnirs.verbose = FALSE)`.

## Details

This function uses `replace_missing()` (based on `stats::approx()`) to interpolate across new samples in the resampled data range.

### Sample rate and time channel:

`time_channel` and `sample_rate` are retrieved automatically from data of class `"mnirs"` which has been processed with `{mnirs}`, if not defined explicitly.

Otherwise, `sample_rate` will be estimated from the values in `time_channel`. However, this may return unexpected values, and it is safer to define `sample_rate` explicitly.

### Default behaviour:

When `resample_rate` is omitted, the output has the same `sample_rate` as the input but with a regular, evenly-spaced `time_channel`. This is useful for regularising data that contains missing or repeated samples without changing the nominal rate.

### Column handling:

Numeric columns are interpolated according to method (see `?replace_missing`). Non-numeric columns (character event labels, integer lap numbers) are always filled by last-observation-carried-forward, regardless of method:

- When down-sampling, the first non-NA value in each output bin is used.
- When up-sampling or regularising, the most recent original value is carried forward into new samples.
- For `method = "none"`, existing rows are matched to the nearest original values of `time_channel` without interpolation or filling, meaning newly created samples and any NAs in the original data are returned as NA.

## Value

A `tibble` of class `"mnirs"`. Metadata are stored as attributes and can be accessed with `attributes(data)`.

## Examples

```
## read example data
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2 = "SmO2 Live"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = TRUE
)

## note warning about irregular sampling
data

data_resampled <- resample_mnirs(
  data,
```

```

    resample_rate = 2, ## blank channels will be retrieved from metadata
    method = "linear", ## blank by default will resample to `sample_rate`
    verbose = TRUE     ## linear interpolation across resampled indices
  )

  ## note the altered `time` values resolving the above warning
  data_resampled

```

---

rescale_mnirs	<i>Re-scale data range</i>
---------------	----------------------------

---

### Description

Expand or reduce the range (min and max values) of data channels to a new amplitude/dynamic range, e.g. re-scale the range of NIRS data to  $c(0, 100)$ .

### Usage

```

rescale_mnirs(
  data,
  nirs_channels = list(NULL),
  range,
  verbose = TRUE
)

```

### Arguments

data	A data frame of class "mnirs" containing time series data and metadata.
nirs_channels	A list() of character vectors indicating grouping structure of mNIRS channel names to operate on (see <i>Details</i> ). Must match column names in data exactly. Retrieved from metadata if not defined explicitly.  list("A", "B", "C") Will operate on each channel independently, losing the relative scaling between channels. list(c("A", "B", "C")) Will operate on all channels together, preserving the relative scaling between channels. list(c("A", "B"), c("C", "D")) Will operate on channels A & B in one group, and C & D in another group, preserving relative scaling within, but not between groups.
range	A numeric vector in the form $c(\min, \max)$ , indicating the range of output values to which data channels will be re-scaled.
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via <code>options(mnirs.verbose = FALSE)</code> .

## Details

`nirs_channels = list()` can be used to group data channels (column names) to preserve absolute or relative scaling.

- Channels grouped together in a vector (e.g. `list(c("A", "B"))`) will be re-scaled to a common range, and the relative scaling within that group will be preserved.
- Channels in separate list vectors (e.g. `list("A", "B")`) will be re-scaled independently, and relative scaling between groups will be lost.
- A single vector of channel names (e.g. `c("A", "B")`) will group channels together.
- Channels (columns) in data not explicitly defined in `nirs_channels` will be passed through untouched to the output data frame.

`nirs_channels` can be retrieved automatically from data of class `"mnirs"` which has been processed with `{mnirs}`, if not defined explicitly. This will default to returning all `nirs_channels` grouped together, and should be defined explicitly for other grouping arrangements.

## Value

A [tibble](#) of class `"mnirs"` with metadata available with `attributes()`.

## Examples

```
## read example data
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2_left = "SmO2 Live",
                   smo2_right = "SmO2 Live(2)"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
) |>
  rescale_mnirs(
    ## un-grouped nirs channels to rescale separately
    nirs_channels = list(smo2_left, smo2_right),
    range = c(0, 100) ## rescale to a 0-100% functional exercise range
  )

data

if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(data, time_labels = TRUE) +
    ggplot2::geom_hline(yintercept = c(0, 100), linetype = "dotted")
}
```

---

scale\_colour\_mnirs      *Scales for custom mnirs palette*

---

### Description

Scales for custom *mnirs* palette

### Usage

```
scale_colour_mnirs(..., aesthetics = "colour")
```

```
scale_color_mnirs(..., aesthetics = "colour")
```

```
scale_fill_mnirs(..., aesthetics = "fill")
```

### Arguments

...                    Arguments passed to `ggplot2::discrete_scale()`.

aesthetics            A character vector with aesthetic(s) passed to `ggplot2::discrete_scale()`.  
*Default* is "colour".

### Value

A `ggplot2` scale object.

### See Also

[theme\\_mnirs\(\)](#), [palette\\_mnirs\(\)](#)

### Examples

```
## plot example data
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2_left = "SmO2 Live",
                   smo2_right = "SmO2 Live(2)"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
)

ggplot2::ggplot(data, ggplot2::aes(x = time)) +
  theme_mnirs() +
  scale_colour_mnirs(name = NULL) +
  ggplot2::geom_line(ggplot2::aes(y = smo2_left, colour = "smo2_left")) +
  ggplot2::geom_line(ggplot2::aes(y = smo2_right, colour = "smo2_right"))
```

---

 shift\_mnirs

*Shift data range*


---

### Description

Move the range of data channels in a data frame up or down, while preserving the absolute amplitude/dynamic range of each channel, and the relative scaling across channels. e.g. shift the minimum data value to zero for all positive values, or shift the mean of the first time span in a recording to zero.

### Usage

```
shift_mnirs(
  data,
  nirs_channels = list(NULL),
  time_channel = NULL,
  to = NULL,
  by = NULL,
  width = NULL,
  span = NULL,
  position = c("min", "max", "first"),
  verbose = TRUE
)
```

### Arguments

data	A data frame of class "mnirs" containing time series data and metadata.
nirs_channels	A character vector giving the names of mNIRS columns to operate on. Must match column names in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the nirs_channels metadata attribute of data is used.</li> </ul>
time_channel	A character string giving the name of the time or sample column. Must match a column name in data exactly. <ul style="list-style-type: none"> <li>• If NULL (default), the time_channel metadata attribute of data is used.</li> </ul>
to	A numeric value in units of nirs_channels to which the data channels will be shifted, e.g. shift the minimum value to zero.
by	A numeric value in units of nirs_channels by which the data channels will be shifted, e.g. shift all values up by 10 units.
width	An integer defining the local window in number of samples centred on idx, between $[idx - \text{floor}(\text{width}/2), idx + \text{floor}(\text{width}/2)]$ .
span	A numeric value defining the local window timespan around idx in units of time_channel or t, between $[t - \text{span}/2, t + \text{span}/2]$ .
position	Indicates where the reference values will be shifted from. <p>"min" (The <i>default</i>) will shift the minimum value(s) to or by the specified value.</p>

	"max" Will shift the maximum value(s) to or by the specified values.
	"first" Will shift first value(s) to or by the specified values.
verbose	Logical. Default is TRUE. Will display or silence (if FALSE) warnings and information messages helpful for troubleshooting. A global default can be set via <code>options(mnirs.verbose = FALSE)</code> .

## Details

`nirs_channels = list()` can be used to group data channels (column names) to preserve absolute or relative scaling.

- Channels grouped together in a vector (e.g. `list(c("A", "B"))`) will be shifted to a common value, and the relative scaling within that group will be preserved.
- Channels in separate list vectors (e.g. `list("A", "B")`) will be shifted independently, and relative scaling between groups will be lost.
- A single vector of channel names (e.g. `c("A", "B")`) will group channels together.
- Channels (columns) in data not explicitly defined in `nirs_channels` will be passed through untouched to the output data frame.

`nirs_channels` and `time_channel` can be retrieved automatically from data of class `"mnirs"` which has been processed with `{mnirs}`, if not defined explicitly. This will default to returning all `nirs_channels` grouped together, and should be defined explicitly for other grouping arrangements.

Only one of either `to` or `by` and one of either `width` or `span` should be defined. If both of either pairing are defined, `to` will be preferred over `by`, and `width` will be preferred over `span`.

## Value

A [tibble](#) of class `"mnirs"` with metadata available with `attributes()`.

## Examples

```
## read example data
data <- read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2_left = "SmO2 Live",
                   smo2_right = "SmO2 Live(2)"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
) |>
  shift_mnirs(
    ## un-grouped nirs channels to shift separately
    nirs_channels = list(smo2_left, smo2_right),
    to = 0,
    ## NIRS values will be shifted to zero
    span = 120,
    ## shift the *first* 120 sec of data to zero
    position = "first"
  )
data
```

```

if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(data, time_labels = TRUE) +
    ggplot2::geom_hline(yintercept = 0, linetype = "dotted")
}

```

---

 theme\_mnirs

*Custom mnirs ggplot2 theme*


---

## Description

A [ggplot2][ggplot2::ggplot2-package] theme for display.

## Usage

```

theme_mnirs(
  base_size = 14,
  base_family = "sans",
  border = c("partial", "full"),
  ink = "black",
  paper = "white",
  accent = "#0080ff",
  ...
)

```

## Arguments

base_size	Base font size, given in pts.
base_family	Base font family.
border	Define either a <i>partial</i> or <i>full</i> border around plots.
ink	Colour for text and lines. <i>Default</i> is "black".
paper	Background colour. <i>Default</i> is "white".
accent	Accent colour for highlights. <i>Default</i> is "#0080ff".
...	Additional arguments to add to [ggplot2::theme()].

## Details

- axis.title = element\_text(face = "bold") by *default* Modify to "plain".
- panel.grid.major & panel.grid.minor set to blank. Modify to = element\_line() for visible grid lines.
- legend.position = "top" by *default* Modify "none" to remove legend entirely.
- border = "partial" uses panel.border = element\_blank() and axis.line = element\_line().
- border = "full" uses panel.border = element\_rect(colour = "black", linewidth = 1) and axis.line = element\_line().
- base\_family = "sans" by *default*.

**Value**

A `ggplot2` theme object.

**See Also**

`palette_mnirs()`, `scale_colour_mnirs()`

**Examples**

```
## plot example data
read_mnirs(
  file_path = example_mnirs("moxy_ramp"),
  nirs_channels = c(smo2_left = "SmO2 Live",
                   smo2_right = "SmO2 Live(2)"),
  time_channel = c(time = "hh:mm:ss"),
  verbose = FALSE
) |>
  plot(time_labels = TRUE)
```

---

train.red\_intervals.csv

*10 Hz Train.Red App export*

---

**Description**

Exported from Train.Red app, recorded at 10 Hz. Containing two 5-minute cycling work intervals, placed on bilateral vastus lateralis muscle sites. Some data channels have been omitted to reduce file size.

**Format**

.csv file with header metadata and 10 columns and 12000 rows:

**Timestamp (seconds passed)** Elapsed time (s).

**Lap/Event** Lap number (numeric).

**SmO2** Muscle oxygen saturation, filtered (%). Two channels have duplicated names. If both are called, the second will be renamed to SmO2\_1.

**SmO2 unfiltered** Muscle oxygen saturation, raw signal (%). Two channels have duplicated names. If both are called, the second will be renamed to SmO2\_unfiltered\_1.

**O2HB unfiltered** Oxyhaemoglobin concentration, raw signal (arbitrary units). Two channels have duplicated names. If both are called, the second will be renamed to O2HB\_unfiltered\_1.

**HHB unfiltered** Deoxyhaemoglobin concentration, raw signal (arbitrary units). Two channels have duplicated names. If both are called, the second will be renamed to HHb\_unfiltered\_1.

Channel mapping for `read_mnirs()`:

- `nirs_channels = c("SmO2", "SmO2 unfiltered", "O2HB unfiltered", "HHb unfiltered")`
- `time_channel = c("Timestamp (seconds passed)")`
- `event_channel = c("Lap/Event")`
- `interval_times = list(start = c(2150.09, 2865.05), end = c(2441.06, 3168.08))`
- `interval_times = list(start = c(65.94, 780.90), end = c(356.91, 1183.93))` from `zero_time`

**Source**

Train.Red (Train.Red B.V.), exported via Train.Red app (<https://train.red/>)

**See Also**

[read\\_mnirs\(\)](#), [example\\_mnirs\(\)](#)

**Examples**

```
example_mnirs("train.red")
```

# Index

artinis\_intervals.xlsx, 2

breaks\_timespan, 3  
by\_label (by\_time), 4  
by\_label(), 4, 8, 9  
by\_lap (by\_time), 4  
by\_lap(), 4, 8, 9  
by\_sample (by\_time), 4  
by\_sample(), 4, 8, 9  
by\_time, 4  
by\_time(), 4, 8, 9

create\_mnirs\_data, 5  
create\_mnirs\_data(), 22

example\_mnirs, 7  
example\_mnirs(), 3, 20, 23, 40  
extract\_intervals, 7  
extract\_intervals(), 4, 5

filter\_butter, 11  
filter\_butter(), 17  
filter\_ma, 13  
filter\_mnirs, 15  
filter\_moving\_average (filter\_ma), 13  
format\_hmmss, 18  
format\_hmmss(), 22

ggplot2, 22, 35, 39  
ggplot2::scale\_x\_time(), 18

moxy\_intervals.csv, 19  
moxy\_ramp.xlsx, 20

palette\_mnirs, 21  
palette\_mnirs(), 35, 39  
plot.mnirs, 22  
portamon-oxcap.xlsx, 23

read\_mnirs, 24  
read\_mnirs(), 3, 19, 20, 23, 39, 40

replace\_invalid (replace\_mnirs), 26  
replace\_missing (replace\_mnirs), 26  
replace\_missing(), 29, 32  
replace\_mnirs, 26  
replace\_outliers (replace\_mnirs), 26  
resample\_mnirs, 31  
rescale\_mnirs, 33

scale\_color\_mnirs (scale\_colour\_mnirs), 35  
scale\_colour\_mnirs, 35  
scale\_colour\_mnirs(), 21, 39  
scale\_fill\_mnirs (scale\_colour\_mnirs), 35

scales::breaks\_timespan(), 3  
shift\_mnirs, 36  
signal::butter(), 11, 12, 16  
signal::filtfilt(), 11, 12, 16  
stats::approx(), 28, 29, 31, 32  
stats::smooth.spline(), 16

theme\_mnirs, 38  
theme\_mnirs(), 21, 35  
tibble, 6, 17, 26, 29, 32, 34, 37  
tibbles, 10  
train.red\_intervals.csv, 39