Package 'readNSx'

November 5, 2026

Title Read 'Blackrock-Microsystems' Files ('NEV', 'NSx')

Version 0.2.0

Description Loads 'Blackrock' <https://blackrockneurotech.com> neural signal data files into the memory, provides utility tools to extract the data into common formats such as plain-text 'tsv' and 'HDF5'.

License MPL-2.0 | file LICENSE

Language en-US

Encoding UTF-8

Imports data.table, fastmap, jsonlite, R6

LinkingTo cpp11, hdf5lib

Suggests bit64, tools, testthat (>= 3.0.0), knitr, rmarkdown, spelling

SystemRequirements little-endian platform

NeedsCompilation yes

Copyright For the readNSx package: Zhengjia Wang.

URL https://dipterix.org/readNSx/, https://github.com/dipterix/readNSx

BugReports https://github.com/dipterix/readNSx/issues

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Author Zhengjia Wang [aut, cre]

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2026-11-05 05:00:12 UTC

Contents

get_channel . . . 2
get_event . . . 3
get_file_type . . . 4
get_nev . . . 4
get_nsp . . . 5
get_nsx . . . 5
get_specification . . . 6
import_nsp . . . 7
read_bci2000 . . . 9
read_nev_header . . . 10

Index **11**

`get_channel` *Get channel data*

Description

Obtain channel information and data from given prefix and channel ID.

Usage

```
get_channel(x, channel_id)
```

Arguments

`x`	path `prefix` specified in `import_nsp`, or `'nev/nsx'` object
`channel_id`	integer channel number. Please be aware that channel number, channel ID, electrode ID refer to the same concept in 'Blackrock' 'NEV' specifications. Electrodes are not physical metals, they refer to channels for historical reasons.

Value

A list containing channel data and meta information, along with the enclosing 'NSx' information; for invalid channel ID, this function returns `NULL`

`get_event` *Get event data packets from 'NEV'*

Description

Get event data packets from 'NEV'

Usage

```
get_event(x, event_type, ...)
```

Arguments

`x`	path `prefix` (see `import_nsp`), or `'nev/nsx'` object
`event_type`	event type to load, common event types are
	`'digital_inputs'` packet identifier 0
	`'spike'` packet identifier 1 to 10000 as of version 3.0
	`'recording'` packet identifier 65529 as of version 3.0, available after version 3.0
	`'configuration'` packet identifier 65530 as of version 3.0, available after version 3.0
	`'log'` packet identifier 65531 as of version 3.0, available after version 3.0
	`'button_trigger'` packet identifier 65532 as of version 3.0, available after version 3.0
	`'tracking'` packet identifier 65533 as of version 3.0, available after version 3.0
	`'video_sync'` packet identifier 65534 as of version 3.0, available after version 3.0
	`'comment'` packet identifier 65535 as of version 3.0, available after version 2.3
`...`	pass to other methods

Value

A data frame of corresponding event type, or `NULL` if event is not found or invalid. For `'spike'`, the data sets of `waveforms.h5`: `channelID`, `timeInSeconds`, `classification`, and `waveform`, a matrix with the waveform of each spike in micro-volts, one column per spike

get_file_type *Get 'Blackrock' file type*

Description

Reads the first 10 bytes containing file type and version information.

Usage

```
get_file_type(path)
```

Arguments

path	path to the 'Blackrock' '.nev' or '.nsx' file, or a binary connection.

Value

A list containing file information, including file type, version information, and normalized absolute path.

get_nev *Load 'NEV' information from path prefix*

Description

Load 'NEV' information from path prefix

Usage

```
get_nev(x, ...)
```

Arguments

x	path prefix specified in import_nsp, or 'nev/nsx' object
...	reserved for future use

Value

'NEV' header information if x is valid, otherwise NULL. See Section "'NEV' Data" in get_specification

get_nsp *Get a collection list containing 'NEV' and 'NSx' headers*

Description

Get a collection list containing 'NEV' and 'NSx' headers

Usage

```
get_nsp(x)
```

Arguments

x	path `prefix` specified in `import_nsp`, or `'nev/nsx'` object

Value

A list containing `'nev'` and imported `'nsx'` headers, see `import_nsp` for details

get_nsx *Load 'NSx' information from path prefix*

Description

Load 'NSx' information from path prefix

Usage

```
get_nsx(x, which, ...)
```

Arguments

x	path `prefix` specified in `import_nsp`, or `'nev/nsx'` object
which	which 'NSx' to load, for example, `which=3` loads `'ns3'` headers
...	reserved for future use

Value

'NSx' header information if data is found, otherwise returns `NULL`. See Section "'NSx' Data" in `get_specification`

`get_specification` *Get '.nev' or 'nsx' specification*

Description

Get '.nev' or 'nsx' specification

Usage

```
get_specification(version, type = c("nev", "nsx"))
```

Arguments

`version`	either character string or a vector of two integers; for example, `"2.2"`, `"2.3"`, `c(3, 0)`. Currently only these three versions are supported since I was unable to find any other versions. Please file an issue ticket if others versions are desired.
`type`	file type; choices are `'nev'` or `'nsx'`

Value

The file specification as a list. The specification usually contains three sections: basic header (fixed length), extended header (dictionary-style), and data packets (data stream). The specification is used to parse the data files.

'NEV' Data

A 'NEV' object consists of three sections:

Section 1 contains basic information such as the time-origin of all the time-stamps, the time-stamp sampling frequency, data packets sizes.

Section 2 is extended header containing the configurations of channels, digital signals, etc. For any data packets in section 3, there should be at least one table in this section describing the settings.

section 3 is a collection of event packets such as digital signal inputs (most likely to be used at version 2.2 or by 'Ripple'), spike waveform, comments (sometimes storing epoch information), etc.

Please be aware that while most common entries can be found across different file versions, some entries are version-specific. If you are making your script general, you need to be very careful handling these differences. For more information, please search for the data specification manual from the 'Blackrock-Microsystems' website.

'NSx' Data

A 'NSx' file refers to the data files ending with `'ns1'` through `'ns9'`. Common types are `'ns2'` (sampling at 1000 Hz), `'ns3'` (sampling at 2000 Hz), and `'ns5'` (sampling at 30,000 Hz).

A 'NSx' file also consists of three sections. Section 1 contains basic information such as the time-origin of all the time-stamps, sampling frequencies, and channel counts within the file. Please be careful that item `time_resolution_timestamp` is not the sampling frequency for signals. This

item is the frequency of the clock that counts time-stamps: usually `30,000`, or `1e9` (nanoseconds) in files recorded with precision time protocol (`PTP`) time-stamps. A time-stamp in seconds is the time-stamp divided by `time_resolution_timestamp`. The signal sample rate is `30000` divided by `period`, the number of `1/30000` seconds between data points. For example, `'ns3'` usually has `period=15`, hence the signal sample rate is `30000/15=2000Hz`.

Section 2 usually contains one and only one channel table of which the number of rows should coincide with number of channels from section 1. Other information such as channel labels, physical connectors, pins, units, filter settings, digital-to-analog conversion are also included. Since `readNSx` always attempts to convert signals in 'volts' or 'milli-volts' to 'micro-volts', the `'units'` column might be different to what's actual recorded in the 'NSx' file headers.

Section 3 contains partitions of continuous recording. Usually each data block (a time-stamp followed by many data points) is one partition; a new data block starts after every pause. In files with `PTP` time-stamps, every data block holds one data point, and consecutive blocks form one partition until a pause, detected from the time-stamps as the 'Blackrock' reader `NPMK` `openNSx` does: options `'readNSx.packets_per_frame'` (default `100000`) and `'readNSx.max_tick_multiple'` (default `2`, must be greater than `1`) correspond to its `packets_per_frame` and `max_tick_multiple` arguments. When imported/loaded from `readNSx`, the digital signals are always converted to analog signals with 'micro-volts' unit. Please use `get_channel` to get the channel data.

Examples

```
get_specification(version = c(2,3), type = "nev")

get_specification(version = "3.0", type = "nsx")
```

`import_nsp` *Import signal data from 'Blackrock-Microsystems' data files*

Description

Please use `import_nsp` to import 'NEV' and 'NSx' files.

Usage

```
import_nsp(
  path,
  prefix = NULL,
  exclude_events = "spike",
  exclude_nsx = NULL,
  verbose = TRUE,
  partition_prefix = "/part"
)
```

Arguments

path	path to 'NEV' or 'NSx' files
prefix	path prefix to save data files into
exclude_events	exclude one or more 'NEV' data events, choices are 'spike', 'video_sync', 'digital_inputs', 'tracking', 'button_trigger', 'comment', 'configuration'; default is 'spike' since the spike data takes very long time to parse, and most users might still use their own offline spike-sorting algorithms.
exclude_nsx	excluded 'NSx' types, integer vectors from 1 to 9; for example, exclude_nsx=c(1,2) will prevent 'ns1' and 'ns2' from being imported
verbose	logical or a progress object: when logical, verbose indicates whether to print the progress as messages; when verbose is a progress object, this object must have inc method; examples are Progress in the shiny package, progress2 from dipsaus, or shiny_progress from shidashi
partition_prefix	additional prefix to the data partition; default is "/part"

Value

A list of configurations, see get_specification for what's contained.

Storage format

Channel signals are written as HDF5 datasets. **readNSx** reads and writes them with its own reader and writer, linked against **hdf5lib**, which ships a static build of the HDF5 library; no system HDF5 installation is required on any platform.

The layout on disk is the conventional one, so exported files can also be read with **hdf5r**, 'h5py', or the HDF5 command-line tools, and files exported by earlier versions of **readNSx** are read back unchanged.

Examples

```
# Please get your own sample data first. This package does not
# provide sample data for privacy and license concerns :)

if(interactive() && file.exists("sampledata.nev")) {

library(readNSx)

# ---- Import for the first time --------------------------------
import_nsp(
  path = "sampledata.nev",
  prefix = file.path(
    "~/BIDSRoot/MyDataSet/sub-YAB/ses-008/ieeg/",
    "sub-YAB_ses-008_task-congruency_acq-NSP1_run-01"
  ),
  exclude_events = "spike", partition_prefix = "/part"
)
```

```
# ---- Load header information --------------------------------
prefix <- "sub-YAB_ses-008_task-congruency_acq-NSP1_run-01"
nev <- get_nev(prefix)
ns3 <- get_nsx(prefix, which = 3)

# get nev from nsx, or nsx from nev
get_nev(ns3)
get_nsx(nev, which = 5)

# ---- Load channel data
result <- get_channel(prefix, channel_id = 10)
channel_signal <- result$channel_detail$part1$data
channel_signal[]

}
```

`read_bci2000` *Read 'BCI2000' recording data*

Description

Read 'BCI2000' recording data

Usage

```
read_bci2000_header(file)

read_bci2000(file)
```

Arguments

`file`	path to the recording data

Value

Parsed signal data

Examples

```
# Package comes with sample data
file <- system.file("samples", "bci2000_sample.dat", package = "readNSx")
result <- read_bci2000(file)

print(result)

# Notive: v1.0 and v1.1 are different, but all in `Source` section
# sample rate
result$parameters$Source$SamplingRate$value
```

```
# Signal data 64 channels x 500 time-points
dim(result$signals)
```

`read_nev_header` *Read 'NEV' headers*

Description

Reads the basic and extended headers of a 'NEV' file, such as the channel labels and waveform settings, without reading the data packets and without writing any file. Unlike `import_nsp`, the time needed does not grow with the length of the recording.

Usage

```
read_nev_header(path)
```

Arguments

`path`	path to the 'NEV' file

Value

A list of `header_basic` (file settings such as the time resolution and the data packet size) and `header_extended` (one data frame per extended header type, for example `NEUEVLBL` for channel labels and `NEUEVWAV` for waveform settings). See `get_specification` for the fields.

Examples

```
# Please get your own sample data first. This package does not
# provide sample data for privacy and license concerns :)

if(interactive() && file.exists("sampledata.nev")) {

header <- read_nev_header("sampledata.nev")

# channel labels
header$header_extended$NEUEVLBL

}
```

Index

get_channel, 2, *7*
get_event, 3
get_file_type, 4
get_nev, 4
get_nsp, 5
get_nsx, 5
get_specification, *4, 5*, 6, *8*, *10*

import_nsp, *2–5*, 7, *10*

read_bci2000, 9
read_bci2000_header *(*read_bci2000*)*, 9
read_nev_header, 10