Package: rYoctoPuceInOut (via r-universe)

May 28, 2026

Type Package

Title Logged Data File Import for YoctoPuce USB Modules

Version 0.0.1.9001

Date 2026-02-08

Description Read CSV files from the data loggers built into YoctoPuce USB sensor modules and JSON files with module settings and other metadata.

License GPL (>= 2)

Depends R (>= 4.1.0)

Encoding UTF-8

Imports utils, jsonlite, photobiology (> 0.14.1), SunCalcMeeus, tidyr, lubridate

Suggests repr, photobiologyLamps, photobiologyLEDs, photobiologySensors (>= 0.5.2), photobiologySun, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

LazyLoad yes

LazyData yes

ByteCompile true

URL https://docs.r4photobiology.info/rYoctoPuceInOut/, https://github.com/aphalo/rYoctoPuceInOut

BugReports https://github.com/aphalo/rYoctoPuceInOut/issues

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

Repository https://aphalo.r-universe.dev

Date/Publication 2026-04-28 09:41:18 UTC

RemoteUrl https://github.com/aphalo/rYoctoPuceInOut

RemoteRef HEAD

RemoteSha dcb11f3c7d7bf3c49a1f83cae12d3b01ea189746

Contents

`calc_calibrated_qtys` *Compute measured quantities*

Description

Compute derived quantities from data logged by a YoctoPuce USB module.

Usage

```
calc_calibrated_qtys(x, calibrations, save.calibrations = TRUE)
```

Arguments

`x`	data.frame as returned by `read_yocto_datalog()` or `read_yocto_spctlog()`.
`calibrations`	list A nested named list with calibration coefficients or function definitions.
`save.calibrations`	logical Flag indicating if the calibration object is to be stored as an attribute in the returned object.

Details

Function `calc_calibrated_qtys()` computes derived quantities from data logged by a YoctoPuce USB module previously imported into R by means of function `read_yocto_datalog()` or function `read_yocto_spctlog()`.

Multiple approaches to calibration are supported, one based on multipliers applied to data for one or more channels and combined by summation. This approach is suitable when the calibration function is obtained by fitting a multiple regression linear model (i.e., with no interaction terms). In this case the calibration object is a `list` where `numeric` values are stored. The name of the members of the calibration object must match the names of the columns of data frame `x`.

The second more flexible approach, relies on user-defined functions stored in the calibration object instead of `numeric` values. In this case the `function` definitions can replace one of more of the

numeric values, allowing more complex conversion computations than a simple multiplication factor. These functions should accept as input a single numeric vector, i.e., the raw data values in one column of x, i.e., data acquired from a single sensor channel and should return a numeric vector of calibrated values of the same length. In this case, as when using numeric multipliers, the values computed from different channels for a single output are added up.

A final variation is a calibration based on a single function that accepts as input a data frame x in whole and returns a vector of calibrated values of the same length as rows in x. This approach is the most flexible and is agnostic about the class of the data in x columns or returned values, i.e., can apply transformations to non-numeric data.

All three approaches can be combined in a single calibration object, with different approaches used for the different derived quantities. This allows using the most suitable/convenient approach for the computation of each derived quantity.

The number of quantities in the returned data frame and their names are determined by the named values stored in the list object calibrations.

Functions read_yocto_datalog() and read_yocto_spctlog() store metadata as attributes in the data frames they return. These attributes are copied to the data frames of calibrated data, adding a trace of the application of calibrations.

Value

A data frame with column "time" with the times from x$time and one column of calibrated data for each primary branch in in the tree-like list object calibrations. The calibrations object is stored as attribute "yocto.module.calibrations" together with those attributes supported by the R for photobiology suite or comment present in x.

Calibration object

The coefficients or functions are stored in a tree-like named list of named lists. The names in the root list are names of the derived quantities while the terminal branches are named for the columns in x, columns each containing raw data for a single channel of a YoctoPuce USB module.

As the names of the columns in x not only depend on the module type but also on module settings and possible renaming during import into R, *calibration objects are data workflow specific*.

Examples

```
# example calibration list based on parallel measurements of sunlight
# with all branches except one relying on numeric multipliers applied to
# a single sensor channel, and one channel relying on the sum of scaled
# values from two sensor channels.

s01.cal <-
  list(module.type = "YoctoSpectral",
       module.sn = "SPECTRL1-2CF3B6",
       module.name = "spectrl-01",
       Q_PAR = list(VIS.avg = 1180 / 9296),
       Q_ePAR = list(VIS.avg = 1377 / 9296),
       Q_e = list(F8.avg = (1377 - 1180) / 3422),
       Q_FarRed = list(F8.avg = 197.87 / 3422),
       Q_Red = list(F6.avg = 265.15 / 5340 * (5340 / (5340 + 6462)),
```

```
                          F7.avg = 265.15 / 6462 * (6462 / (5340 + 6462))))

# path to example logged data from a Yocto-Spectral USN module
yocto_spectral.file <-
  system.file("extdata", "yocto-spectral-LED.csv",
              package = "rYoctoPuceInOut", mustWork = TRUE)
yocto_spectral_json.file <-
  system.file("extdata", "SPECTRL1-2CF3B6.json",
              package = "rYoctoPuceInOut", mustWork = TRUE)

# read data and apply calibration
read_yocto_spctlog(yocto_spectral.file) |>
  calc_calibrated_qtys(s01.cal)

read_yocto_spctlog(yocto_spectral.file, yocto_spectral_json.file) |>
  calc_calibrated_qtys(s01.cal)
```

`read_yocto_datalog` *Read data from YoctoPuce CSV files*

Description

Functions able to directly read and validate data from .CSV files created by the data loggers built into YoctoPuce USB modules, and metadata extracted from a JSON file with the USB module settings.

Usage

```
read_yocto_datalog(
  file,
  settings.file = NULL,
  geocode = NULL,
  label = NULL,
  cols.pattern = NULL,
  cols.logical.names = FALSE,
  nacols.rm = TRUE,
  dec = ".",
  sep = ";",
  tz = "UTC",
  module.descriptor = NULL,
  ...
)

read_yocto_spctlog(
  file,
  settings.file = NULL,
  cols.pattern = "avg",
```

```
  cols.rename = TRUE,
  dec = ".",
  sep = ";",
  tz = "UTC",
  ...
)
```

Arguments

file	the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, getwd(). Tilde-expansion is performed where supported. This can be a compressed file (see file). Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.) file can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for url.)
settings.file	character Path to a JSON file containing the module settings used to acquire the data.
geocode	data.frame Containing columns lon and lat used to set attribute "where.measured".
label	character Additional text to be appended to the default value used to set attribute "comment".
cols.pattern	character A string suitable as argument for pattern in a call to grep() on column headings in the imported CSV file. NULL or characte(0) retain all data columns, while NA returns all data and time columns.
cols.logical.names	logical Use logical names from module metadata instead of column heading in CSV file. Require readable settings.file.
nacols.rm	logical If TRUE delete columns that contain only NA values.
dec	the character used in the file for decimal points.
sep	the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.
tz	a character string that specifies which time zone to parse the date with. The string must be a time zone that is recognized by the user's OS.
module.descriptor	list A matadata descriptor of the YocotoPuce USB module to be added as an attribute to the returned object.
...	Arguments passed on to utils::read.table colClasses character. A vector of classes to be assumed for the columns. If unnamed, recycled as necessary. If named, names are matched with unspecified values being taken to be NA.

Possible values are NA (the default, when type.convert is used), "NULL" (when the column is skipped), one of the atomic vector classes (logical, integer, numeric, complex, character, raw), or "factor", "Date" or "POSIXct". Otherwise there needs to be an as method (from package **methods**) for conversion from "character" to the specified formal class.
Note that colClasses is specified per column (not per variable) and so includes the column of row names (if any).

nrows integer: the maximum number of rows to read in. Negative and other invalid values are ignored.

skip integer: the number of lines of the data file to skip before beginning to read data.

check.names logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

comment.char character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.

cols.rename logical Flag, if TRUE use channel names from sensor IC specification as column names, and if FALSE keep the names used in the imported file.

Details

The dataloggers implemented in different USB modules from YoctoPuce return .CSV files with a consistent format, that varies only in the number of data columns and their names. Function read_yocto_datalog() reads any of these files preserving column names. The UTC times are converted into POSIXct values and added under column time. A subset of columns can be requested by passing a regular expression to be used in a call to grep() on the column names. If nacols.rm = TRUE is passed, columns containing only NA values are deleted, as these columns in most cases correspond to logger channels that are not in use.

Warnings are issued if something unexpected, such as dates several years into the past, unsorted or repeated time stamps are encountered in the sequence of UNIX time stamps in the file. These discrepancies can occur when the modules' or the YoctoHub clocks do not acquire a valid time at start-up. Inconsistent time steps are reported through a message, as these are the result of a stop followed by restart of logging. During such a pause it is possible that module settings could have been changed.

If the name of a JSON file as saved from the YoctoPuce module is passed as an argument to parameter settings.file its contents parsed into an R list are saved to attribute "yocto.module.settings" in the returned data frame. The metadata also make it possible to set column names to the logical names set in the module. The logical name of the module and the units are also extracted if available, formatted and saved in the "how.measured" attribute.

A comment attribute is always set with information about the imported file(s) and the import time and package.

Function read_yocto_spctlog() is a wrapper on read_yocto_datalog() that renames columns using the names used in the data sheet for the AS7343 sensor.

In modules with multiple channels, the user can enable and disable logging on a channel by channel basis. Thus, the number of data columns can vary, making it necessary to match columns by name when replacing some of the default names by shorter or more informative ones.

Value

A data frame with `POSIXct` time stamps in column `time` and data in a variable number of columns containing values converted by `read.csv()`.

Warning!

The units and basis of expression, and in several cases even the physical quantity measured cannot be discovered from the CSV files. The values in the returned data frame are those read from the file, and the read values have to be interpreted based on the module settings. These settings if available in a JSON file downloaded from the module, can be stored in attribute `"yocto.module.settings"` of the data frame returned.

I have access to several different USB modules from YoctoPuce, but not to all of them. There are currently some modules that are not fully supported, and a few others that "should" work but have not been yet tested.

References

Documentation for each YocotoPuce USB module is available at https://www.yoctopuce.com/.

Examples

```
library(photobiology)
# Yocto-Meteo module

yocto_meteo.file <-
  system.file("extdata", "yocto-meteo-snm.csv",
              package = "rYoctoPuceInOut", mustWork = TRUE)
yocto_meteo_json.file <-
  system.file("extdata", "METEOMK2-19A230.json",
              package = "rYoctoPuceInOut", mustWork = TRUE)

meteo1.df <- read_yocto_datalog(yocto_meteo.file)
head(meteo1.df, n = 5)
cat(comment(meteo1.df))
cat(how_measured(meteo1.df))

read_yocto_datalog(yocto_meteo.file, cols.pattern = "avg") |>
  head(n = 5)
read_yocto_datalog(yocto_meteo.file, cols.pattern = "min|max") |>
  head(n = 5)
read_yocto_datalog(yocto_meteo.file, cols.pattern = "temperature") |>
  head(n = 5)
read_yocto_datalog(yocto_meteo.file, nrows = 4L)

# metadata from JSON file
meteo2.df <- read_yocto_datalog(yocto_meteo.file, yocto_meteo_json.file)
```

```
head(meteo2.df, n = 5)
cat(comment(meteo2.df))
cat(how_measured(meteo2.df))

meteo3.df <- read_yocto_datalog(yocto_meteo.file,
                                yocto_meteo_json.file,
                                cols.logical.names = TRUE)
head(meteo3.df, n = 5)
cat(comment(meteo3.df))
cat(how_measured(meteo3.df))
str(attr(meteo3.df, "yocto.module.settings"), max.level = 2)

# Yocto-Spectral module

yocto_spectral.file <-
  system.file("extdata", "yocto-spectral-LED.csv",
              package = "rYoctoPuceInOut", mustWork = TRUE)
yocto_spectral_json.file <-
  system.file("extdata", "SPECTRL1-2CF3B6.json",
              package = "rYoctoPuceInOut", mustWork = TRUE)

read_yocto_spctlog(yocto_spectral.file) |> head(n = 5)
read_yocto_spctlog(yocto_spectral.file, cols.pattern = "Channel1\\.") |>
  head(n = 5)
read_yocto_spctlog(yocto_spectral.file, cols.rename = FALSE) |>
  head(n = 5)

# metadata from JSON file
spectral.df <-
  read_yocto_spctlog(yocto_spectral.file,
                          yocto_spectral_json.file)
cat(how_measured(spectral.df))
cat(comment(spectral.df))
```

`y_spectral.descriptor` *Metadata for YoctoPuce modules*

Description

Metadata for sensors used in YoctoPuce USB modules. The metadata were retrieved from module specifications or the data sheet for the sensor used in the modules. As such, they are *typical* or *nominal* values. Individual sensors' properties are dependent on manufacturing tolerances and on temperature. For accurate measurements sensors need to be individually calibrated keeping in mind that sensors need recalibration as their properties can also change with time.

Usage

```
y_spectral.descriptor
```

Format

An object of class `list` of length 10.

See Also

Package `photobiologySensors-package` provides spectral response data for numerous light sensors relevant to light measurements relevant to plants and vegetation.

Examples

```
names(y_spectral.descriptor)
y_spectral.descriptor$peak.wl
y_spectral.descriptor$HMFW
```

---

yocto_spectral.df *Example of imported spectral data*

Description

A dataset containing data logged with a Yocto-Spectral USB module from YoctoPuce and imported with function `read_yocto_spctlog()`.

Usage

```
yocto_spectral.df
```

Format

A `data.frame` with 199 rows and 14 variables.

See Also

`yocto_spectral2mspct()`, `{read_yocto_datalog}()`.

Examples

```
head(yocto_spectral.df)
```

`yocto_spectral2mspct` *Yocto-Spectral to spectra*

Description

Convert a data frame with data imported with `read_yocto_spctlog()` into a collection of `raw_spct` objects.

Usage

```
yocto_spectral2mspct(df, channels = "all", ...)
```

Arguments

`df`	data.frame With only one of average, minimum or maximum sensor counts for each spectral channels.
`channels`	character vector of length one or longer. In addition to a vector of spectral sensor channel names as used in the AS7343 documentation, the nicknames `"all"`, `"wide"`, `"narrow"` and `"xyz"` as vectors of length one are accepted.
`...`	Arguments passed on to `photobiology::subset2mspct` `drop.idx` logical Flag indicating whether to drop or keep idx.var in the collection members. `ncol` integer Number of 'virtual' columns in data. `byrow` logical If `ncol` > 1 how to read in the data.

Details

The data frame, possibly after subsetting data columns, is pivoted into long form, and subsequently split by time stamp. The readings at each time point are converted into *short* spectra with at most 13 wavelength values, one from each channel.

Note

Currently, function `subset2mspct()` used in the conversion into a `raw_mspct` object can be slow in the case of 10000's of spectra.

See Also

`subset2mspct()`, `raw_spct()`.

Examples

```
nrow(yocto_spectral.df)
comment(yocto_spectral.df)
yocto_spectral.mspct <- yocto_spectral2mspct(yocto_spectral.df)
summary(yocto_spectral.mspct)
yocto_spectral.mspct[[1]]
```

Index