Package: data.io (via r-universe)

July 15, 2024

Type Package

Version 1.5.1

Title Read and Write Data in Different Formats

Description Read or write data from many different formats (tabular datasets, from statistic software ...) into R objects. Add labels and units in different languages.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 4.2.0)

Imports Hmisc (>= 5.0.1), lifecycle (>= 1.0.3), utils (>= 4.2.0), readr (>= 2.1.4), rlang (>= 1.1.1), svBase (>= 1.4.0), tibble (>= 3.2.1), tsibble (>= 1.1.3)

Suggests babynames (>= 1.0.1), datasets (>= 4.2.0), ggplot2 (>= 3.4.2), haven (>= 2.5.2), lme4 (>= 1.1.32), nycflights13 (>= 1.0.2), palmerpenguins (>= 0.1.1), readxl (>= 1.4.2), writexl (>= 1.4.2), WriteXLS (>= 6.4.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

Remotes SciViews/svBase

License MIT + file LICENSE

URL https://github.com/SciViews/data.io, https://www.sciviews.org/data.io/

BugReports https://github.com/SciViews/data.io/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData yes

ByteCompile yes

Config/testthat/edition 3

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

RemoteUrl https://github.com/SciViews/data.io

RemoteRef HEAD

RemoteSha 5b5d92d00288223305f8a2f82a56ac6fe820ec45

Contents

`data.io-package` *Read and Write Data in Different Formats*

Description

The {data.io} package focuses on reading and writing datasets in different formats in an unified and convenient way. It can deal with labels and units metadata for variables, translation in different languages, and even use a sidecar file for preprocessing the dataset automatically. The same features are also available for a subset of datasets from R packages.

Important functions

- `read()` is the main function to read data from R packages or files,
- `write()` is the main function to write data to disk. It is compatible with `base::write()` but provides many more features if you indicate `type=` or use it like `write$type()`.
- `labelise()` adds a `label`, and possibly a `units` attributes to an object, to be used while pretty printing a table or plot.

`as_dataframe` *Deprecated! Convert objects into dataframes (subclassing tibble) and check for it.*

Description

[Deprecated]

Convert an object into a `dataframe` and check for it. A `dataframe` (without dot) is both a `data.frame` (with dot, the default rectangular dataset structure in R) and a `tibble`, the tidyverse equivalence. In fact, `dataframes` behave almost completely like a `tibble`, except for a few details explained in the **details** section.

Usage

```
as_dataframe(x, ...)

as.dataframe(x, ...)

## Default S3 method:
as_dataframe(x, tz = "UTC", ...)

## S3 method for class 'data.frame'
as_dataframe(x, ..., rownames = "rownames")

## S3 method for class 'dataframe'
as_dataframe(
  x,
  ...,
  rownames = "rownames",
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)

## S3 method for class 'list'
as_dataframe(
  x,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  ...
)

## S3 method for class 'matrix'
as_dataframe(x, ..., rownames = "rownames")

## S3 method for class 'table'
as_dataframe(x, n = "n", ...)

is_dataframe(x)
```

```
is.dataframe(x)
```

Arguments

x	An object to convert to a dataframe.
...	Additional parameters.
tz	The time zone. Useful for converting ts objects with observations more frequent than daily.
rownames	Name of the column that is prepended to the dataframe with the original row names (dataframes and tibbles do not support row names). If NULL, row names are dropped. The inclusion of the rownames column is **not** done if row names are trivial, i.e., they equal the number of the rows in the data frame.
.name_repair	Treatment for problematic column names. "check.unique" (default value) do not repair names but make sure they are unique. "unique" make sure names are unique and non empty. "universal" make names unique and syntactic. "minimal"do not repair or check (just make sure names exist).
n	The name for the column containing the number of items, "n" by default.

Details

TODO: explain difference between dataframes and tibbles here...

Value

A dataframe, which is an S3 object with class c("dataframe", "tbl_df", "tbl", "data.frame").

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

as_tibble(), as.data.frame()

Examples

```
class(as.dataframe(mtcars))
class(as.dataframe(tibble::tribble(~x, ~y, 1, 2, 3, 4)))

# Any object, like a vector
v1 <- 1:10
is_dataframe(v1)
(df1 <- as_dataframe(v1))
is_dataframe(df1)
# Check names of an existing dataframe
(as_dataframe(df1, .name_repair = "universal"))
# A data.frame with trivial row names
datasets::iris
```

```
as_dataframe(datasets::iris)
# A data.frame containing meaningful row names
datasets::mtcars
as_dataframe(datasets::mtcars)
# A list
l1 <- list(x = 1:3, y = rnorm(3))
as_dataframe(l1)
# A matrix with column and row names
(m1 <- matrix(1:9, nrow = 3L, dimnames = list(letters[1:3], LETTERS[1:3])))
as_dataframe(m1)
# A table
set.seed(756)
(t1 <- table(sample(letters[1:5], 50, replace = TRUE)))
as_dataframe(t1)
# compare with the base R function:
as.data.frame(t1)
```

`Datasets` *Labelised versions of various datasets provided by 'data.io' or other packages*

Description

Use `name <- read("data", package = "pkg", lang = "xx")` to read these datasets together with the metadata (labels, units, comments, ...).

Details

From `data`:

`mauna_loa` Temperature and atmospheric CO2 at Mauna Loa, Hawai. 5 vars x 768 obs. Time series of monthly averages from 1955 to 2018.

`urchin_bio` Sea urchins biometry. 19 vars x 421 obs. Morphometric variables measured on two populations of sea urchins, incl. one circular variable (`maturity`).

`urchin_growth` Sea urchins growth. 3 vars x 7024 obs. Size at age for a cohort of sea urchins followed over more than 10 years.

`zooplankton` Zooplankton image analysis. 20 vars x 1262 obs. A training set with 19 measurements made on images of zooplankton and their respective class as attributed by taxonomists.

From `datasets`:

`anscombe` Anscombe's quartet of 'identical' simple linear Regressions. 8 vars x 11 obs. Artificial data.

`iris` Edgar Anderson's iris data. 5 vars x 150 obs. Morphometry of the flowers of three iris species (50 for each species).

`lynx` Annual Canadian lynx trappings 1821–1934. 2 vars x 114 obs. Long (> 1 century) time series.

`trees` Black cherry trees measurements. 3 vars x 31 obs. Measurement of tree timber of various sizes.

From `ggplot2`:

ggplot2::diamonds Prices of 50,000 round cut diamonds. 10 vars x 53940 obs. Price and other attributes of 10,000's of diamonds.

ggplot2::mpg Fuel economy data from 1999 and 2008 for popular cars. 11 vars x 234 obs. Data are for most popular U.S. market cars only.

From `MASS`:

`crabs` Morphological measurements on Leptograpsus crabs. 8 vars x 200 obs. Morphological measurements of *Leptograpsus variegatus* crabs, either blue or orange, males and females.

`geyser` Old Faithful geyser data. 2 vars x 299 obs. Duration and waiting time for eruptions from August 1 to August 15, 1985.

From `nycflights13`:

nycflights13::airlines Airlines by their carrier codes. 2 vars x 16 obs.

nycflights13::airports Various metadata about New York city airports. 8 vars x 1458 obs.

nycflights13::flights On-time data for all flights that departed NYC (i.e., JFK, LGA or EWR) in 2013. 19 vars x 336776 obs.

nycflights13::planes Planes metadata. 9 vars x 3322 obs.

nycflights13::weather Hourly meteorological data for JFK, LGA and EWR. 15 vars x 26130 obs.

`data_example` *Get the path to some example datasets in this package*

Description

Get the full path to so example datasets included in different formats in the "data.io" package.

Usage

```
data_example(path)
```

Arguments

`path`	The subpath to a file inside the "extdata" subdirectory of the "data.io" package.

Value

The path to the file, or "" if it is not found.

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

read()

Examples

```
data_example("iris.csv")
```

`data_types` *List recognized file formats (types) for read() and write()*

Description

Display information about data types that can read() and write() can use, as well as, the original functions that are delegated (see they respective help pages for more info and to know which additional parameters can be used in read() and write()).

Usage

```
data_types(types_only = FALSE, view = TRUE)
```

Arguments

`types_only`	If `TRUE`, only a vector of types is returned, otherwise, a `tibble` with full specifications is provided.
`view`	If `TRUE`, the result is "viewed" (displayed in a table in a separate window, if the user interface allows it, e.g., in RStudio) and returned invisibly. Otherwise, the results are returned normally.

Details

The function is mainly designed to be used interactively and to provide information about file types that can be read() or write(). This cannot be done through a man page because this list is dynamic and other packages could add or change entries there. With `view = FALSE`, the function can, nevertheless, be also used in a script or a R Markdown/Notebook document.

Value

An `tibble` with `types_only = FALSE`, or a character vector.

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

read(), write()

Examples

```
## Not run:
data_types()
data_types(TRUE)

## End(Not run)
# For non-interactive use, specify view = FALSE
data_types(view = FALSE)
data_types(TRUE, view = FALSE)
```

`labelise` *Set label (and units)*

Description

Set the `label`, as well as the `units` attributes to an object. The label can be used for better display as plot axes labels, or as table headers in pretty-formatted R outputs. The units are usually associated to the label in axes labels for plots. `cl()` is a shortcut for concatenate (`c()`) and `labelise()`.

Usage

```
labelise(x, label, units = NULL, as_labelled = FALSE, ...)

labelize(x, label, units = NULL, as_labelled = FALSE, ...)

## Default S3 method:
labelise(x, label, units = NULL, as_labelled = FALSE, ...)

## S3 method for class 'data.frame'
labelise(x, label, units = NULL, as_labelled = FALSE, self = TRUE, ...)

cl(..., label = NULL, units = NULL, as_labelled = FALSE)

unlabelise(x, ...)

unlabelize(x, ...)

## Default S3 method:
unlabelise(x, ...)

## S3 method for class 'data.frame'
unlabelise(x, self = TRUE, ...)
```

Arguments

`x`	An object.
`label`	The character string to set as `label` attribute to `x`.

units	The units (optional) as a character string to set for x.
as_labelled	Should the object be converted as a labelled S3 object (no by default)? If you don't make labelled objects, subsetting the data will lead to a lost of label and units attributes for all variables. On the other hand, labelled objects are not always correctly handled by R code.
...	Further arguments: items to be concatenated in a vector using c(...) for cl().
self	Do we label the data.frame itself (self = TRUE, by default) or variables within that data.frame (self = FALSE)? In the later case, label= and units= must be either lists or character vectors of the same length as x, or be named with the names of several or all x variables.

Details

The same mechanism as the one used in package **Hmisc** is used here. However, **Hmisc** always add the **labelled** class to an object, while here, this is optional. Setting this class make the object more nicely printed, and subsettable without loosing these attributes. But it conflicts with a class of the same name in package **haven**, used for other purposes. So, here, one can also opt not to set it, using as_labelled = FALSE.

Value

The x object plus a label attribute, and possibly, a units attribute.

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

label(), units()

Examples

```
# Labelise a vector:
x <- 1:10
x <- labelise(x, label = "A suite of integers", units = "cm")
x
# or, in a single operation:
x <- cl(1:10, label = "A suite of integers", units = "cm")
x
# Not adding the labelled class:
x <- cl(1:10, label = "Integers", units = "cm", as_labelled = FALSE)
x
# Unlabelising a labelised object
unlabelise(x)

# Labelise a data.frame
iris <- labelise(datasets::iris, "The famous iris dataset")
unlabelise(iris)
# but if you indicate self = FALSE, you can labelise variables within the
```

```
# data.frame (use a list or character vector of same length as x, or a
# named list or character vector):
iris <- labelise(iris, self = FALSE, label = list(
  Sepal.Length = "Length of the sepals",
  Petal.Length = "Length of the petals"
  ), units = c(rep("cm", 4), NA))
iris <- unlabelise(iris, self = FALSE)
```

`mauna_loa` *Temperature and atmospheric CO2 at Mauna Loa, Hawai*

Description

Monthly averages of temperatures and CO2 concentrations, maximal and minimal monthly temperatures at Mauna Loa slope observatory from 1955 to 2018.

Usage

```
mauna_loa
```

Format

An object of class `mts` (inherits from `ts`, `matrix`) with 768 rows and 4 columns.

Details

Atmospheric CO2 concentration is mole fraction in dry air, micromol/mol, abbreviated as ppm. Temperatures are in degree Celsius.

Examples

```
class(mauna_loa)
head(mauna_loa)
plot(mauna_loa)

# Using read(), the dataset becomes an annotated dataframe
(ml_en <- read("mauna_loa", package = "data.io"))
class(ml_en)

# Indicating lang = "EN_US" (all uppercase!) also converts temperatures
# into degrees Farenheit
(ml_en_us <- read("mauna_loa", package = "data.io", lang = "EN_US"))
# Each variable is also labelled:
ml_en$avg_co2

# The same in French:
 (ml_fr <- read("mauna_loa", package = "data.io", lang = "fr"))
  ml_fr$avg_co2
```

read *Read data in* R *in different formats*

Description

Read and return an R object from data on disk, from URL, or from packages.

Usage

```
read(
  file,
  type = NULL,
  header = "#",
  header.max = 50L,
  skip = 0L,
  locale = default_locale(),
  lang = getOption("data.io_lang", "en"),
  lang_encoding = "UTF-8",
  as_dataframe = FALSE,
  as_labelled = FALSE,
  comments = NULL,
  package = NULL,
  sidecar_file = TRUE,
  fun_list = NULL,
  hfun = NULL,
  fun = NULL,
  data,
  cache_file = NULL,
  method = "auto",
  quiet = FALSE,
  force = FALSE,
  ...
)

type_from_extension(file, full = FALSE)

hread_text(file, header.max, skip = 0L, locale = default_locale(), ...)

hread_xls(file, header.max, skip = 0L, locale = default_locale(), ...)

hread_xlsx(file, header.max, skip = 0L, locale = default_locale(), ...)

## S3 method for class 'subsettable_type'
x$name

## S3 method for class 'read_function_subset'
.DollarNames(x, pattern = "")
```

Arguments

file	The path to the file to read, or the name of the dataset to get from an R package (in that case, you **must** provide the package= argument).
type	The type (format) of data to read.
header	The character to use for the header and other comments.
header.max	The maximum of lines to consider for the header.
skip	The number of lines to skip at the beginning of the file.
locale	A readr locale object with all the data regarding required to correctly interpret country-related items. The default value matches R defaults as US English + UTF-8 encoding, and it is advised to be used as much as possible.
lang	The language to use (mainly for comment, label and units), but also for factor levels or other character strings if a translation exists and if the language is spelled with uppercase characters (e.g., "FR"). The default value can be set with, e.g., options(data.io_lang = "fr") for French.
lang_encoding	Encoding used by R scripts for translation. They should all be encoded as UTF-8, which is the default. However, this argument allows to specify a different encoding if needed.
as_dataframe	Deprecated: now use options(SciViews.as_dtx = as_XXX) to specify if you want a data.frame (as_dtf), a data.table (as_dtt, by default), or a tibble (as_dtbl). Do we try to convert the resulting object into a dataframe (inheriting from data.frame, tbl and tbl_db alias tibble)? If FALSE, no conversion is attempted. Note that now, whatever you indicate, it is always assumed to be FALSE as part of the deprecation!
as_labelled	Are variable converted into 'labelled' objects. This allows to keep labels and units when the vector is manipulated, but it can lead to incompatibilities with some R code (hence, it is FALSE by default).
comments	Comments to add in the created object.
package	The package where to look for the dataset. If file= is not provided, a list of available datasets in the package is displayed.
sidecar_file	If TRUE and a file with same name as file= + .R is found in the same directory, it is considered as code to import these data and it is sourced with local = TRUE, chdir = TRUE and verbose = FALSE. That script **must** create an object named dataset, which is the result that is returned by the function. It is advised to encode this script in UTF-8, which is the default value, but it is possible to specify a different encoding through the lang_encoding= parameter.
fun_list	The table with correspondence of the types, read, and write functions.
hfun	The function to read the header (lines starting with a special mark, usually '#' at the beginning of the file). This function must have the same arguments as hread_text() and should return a character string with the first header.max lines.
fun	The function to delegate reading of the data. If NULL (default), The function is chosen from fun_list.
data	A synonym to file= (the name makes more sense when the dataset is loaded from a package). You cannot use data= and file= at the same time.

cache_file	The path to a local file to use as a cache when file is downloaded (http://, https://, ftp://, or file:// protocols). If cache_file already exists, data are read from this cache, except if force = TRUE, see here under. Otherwise, data are saved in it before being used. If cache_file = NULL (the default), a temporary file is used and data are read from the Internet every time. This cache mechanism is particularly useful to provide data associated with a git repository. Put cache_file in .gitignore and use cache_file= in the code (and force = FALSE). That way, the data are downloaded once in a freshly cloned repository, and they are not included in the versioning system (useful for large datasets).
method	The downloading method used ("auto" by default), see utils::download.file().
quiet	In case we have to download files, do it silently (TRUE) or do we provide feedback and a progression bar (FALSE, by default)?
force	If TRUE and an URL is provided for file= and a path for cache_file=, then the content is downloaded all the time, even if the cache file already exists (it overwrites it). By default, it is FALSE, which is the most useful setting to make good use of the cache mechanism.
...	Further arguments passed to the function fun=.
full	Do we return the full extension, like csv.tar.gz (TRUE), or only the main extension, like csv (FALSE, by default).
x	A subsettable_type function.
name	The value to use for the type= argument.
pattern	A regular expression to list matching names.

Details

read() allows for a unique entry point to read various kinds of data, but it delegates the actual work to various other functions dispatched across several R packages. See getOption("read_write").

Value

An R object with the data (its class depends on the data being read).

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

data_types(), write(), read_csv()

Examples

```
# Use of read() as a more flexible substitute to data() (can change dataset
# name and syntax more similar to read R datasets and datasets from files)
read() # List all available datasets in your installed version of R
# List datasets in one particular package
read(package = "data.io")
```

```
# Read one dataset from this package, possibly changing its name
(urchin <- read("urchin_bio", package = "data.io"))
# Same, but using labels in French
(urchin <- read("urchin_bio", package = "data.io", lang = "fr"))
# ... and also the levels of factors in French (note: uppercase FR)
(urchin <- read("urchin_bio", package = "data.io", lang = "FR"))

# Read one dataset from another package, but with labels and comments
data(iris) # The R way: you got the initial datasets
# Same result, using read()
ir2 <- read("iris", package = "datasets", lang = NULL)
# ir2 records that it comes from datasets::iris
attr(comment(ir2), "src")
# otherwise, it is identical to iris, except is may be a data.table or a
# tibble, depending on user preferences
comment(ir2) <- NULL
# Force coercion into a data.frame
ir2 <- svBase::as_dtf(ir2)
identical(iris, ir2)
# More interesting: you can get an enhanced version of iris with read():
# (note that variable names ar in snake-case now!)
(ir3 <- read("iris", package = "datasets"))
class(ir3)
comment(ir3)
ir3$sepal_length
# ... and you can get it in French too!
(ir_fr <- read("iris", package = "datasets", lang = "fr"))
class(ir_fr)
comment(ir_fr)
ir_fr$sepal_length

# Sometimes, datasets are more deeply reworked. For instance, trees has
# variables in imperial units (in, ft, and cubic ft), but it is automatically
# reworked by read() into metric variables (m or m^3):
data(trees)
head(trees)
(trees2 <- read("trees", package = "datasets"))
comment(trees2)
trees2$volume

# Read from a Github Gist (need to specify the type here!)
# (ble <- read$csv("http://tinyurl.com/Biostat-Ble"))

# Various versions of the famous iris dataset
(iris <- read(data_example("iris.csv")))
(iris <- read(data_example("iris.csv.zip")))
(iris <- read(data_example("iris.csv.gz")))
(iris <- read(data_example("iris.csv.bz2")))
(iris <- read(data_example("iris.tsv")))
(iris <- read(data_example("iris.xls")))
(iris <- read(data_example("iris.xlsx")))
(iris <- read(data_example("iris.rds"))) # Does not tranform into tibble!
#(iris <- read(data_example("iris.syd"))) ##
```

```
#(iris <- read(data_example("iris.csvy"))) ##
#(iris <- read(data_example("iris.csvy.zip"))) ##

# A file with an header both in English (default) and in French
(iris <- read(data_example("iris_short_header.csv")))
(iris_fr <- read(data_example("iris_short_header.csv"), lang = "fr"))
# Headers are also recognized in xls/xlsx files
(iris_fr <- read(data_example("iris_short_header.xls"), lang = "fr"))

# Read a file with a sidecar file (same name + '.R')
(iris <- read(data_example("iris_sidecar.csv"))) # lang = "en" by default
(iris <- read(data_example("iris_sidecar.csv"), lang = "EN")) # Full lang
(iris <- read(data_example("iris_sidecar.csv"), lang = "en_us")) # US (in)
(iris <- read(data_example("iris_sidecar.csv"), lang = "fr")) # French
(iris <- read(data_example("iris_sidecar.csv"), lang = "FR_BE")) # Belgian
(iris <- read(data_example("iris_sidecar.csv"), lang = NULL)) # No labels

# Require the feather package
#(iris <- read(data_example("iris.feather"))) # Not available for all Win

# Challenging datasets from the readr package
library(readr)
(mtcars <- read(readr_example("mtcars.csv")))
(mtcars <- read(readr_example("mtcars.csv.zip")))
(mtcars <- read(readr_example("mtcars.csv.bz2")))
(challenge <- read(readr_example("challenge.csv"), guess_max = 1001))
(massey <- read(readr_example("massey-rating.txt")))
# By default, the type cannot be guessed from the extension
# This is a space-separated vaules file (ssv)
(massey <- read(readr_example("massey-rating.txt"), type = "ssv"))
# or ...
(massey <- read$ssv(readr_example("massey-rating.txt")))
(epa <- read$ssv(readr_example("epa78.txt"), col_names = FALSE))
(example_log <- read(readr_example("example.log")))
# There are different ways to specify columns for fixed-width files (fwf)
# See ?read_fwf in package readr
(fwf_sample <- read$fwf(readr_example("fwf-sample.txt"),
   col_positions =  fwf_cols(name = 20, state = 10, ssn = 12)))

# Various examples of Excel datasets from readxl
library(readxl)
(xl <- read(readxl_example("datasets.xls")))
(xl <- read(readxl_example("datasets.xlsx"), sheet = "mtcars"))
(xl <- read(readxl_example("datasets.xlsx"), sheet = 3))
# Accomodate a column with disparate types via col_type = "list"
(clip <- read(readxl_example("clippy.xls"), col_types = c("text", "list")))
(clip <- read(readxl_example("clippy.xlsx"), col_types = c("text", "list")))
tibble::deframe(clip)
# Read from a specific range in a sheet
(xl <- read(readxl_example("datasets.xlsx"), range = "mtcars!B1:D5"))
(deaths <- read(readxl_example("deaths.xls"), range = cell_rows(5:15)))
(deaths <- read(readxl_example("deaths.xlsx"), range = cell_rows(5:15)))
(type_me <- read(readxl_example("type-me.xls"), sheet = "logical_coercion",
```

```
  col_types = c("logical", "text")))
(type_me <- read(readxl_example("type-me.xlsx"), sheet = "numeric_coercion",
  col_types = c("numeric", "text")))
(type_me <- read(readxl_example("type-me.xls"), sheet = "date_coercion",
  col_types = c("date", "text")))
(type_me <- read(readxl_example("type-me.xlsx"), sheet = "text_coercion",
  col_types = c("text", "text")))
(xl <- read(readxl_example("geometry.xls"), col_names = FALSE))
(xl <- read(readxl_example("geometry.xlsx"), range = cell_rows(4:8)))

# Various examples from haven
library(haven)
haven_example <- function(path)
  system.file("examples", path, package = "haven", mustWork = TRUE)
(iris2 <- read(haven_example("iris.dta"))) # Stata v. 8-14
(iris2 <- read(haven_example("iris.sav"))) # SPSS, TODO: labelled -> factor?
(pbc <- read(data_example("pbc.por"))) # SPSS, POR format
(iris2 <- read$sas(haven_example("iris.sas7bdat"))) # SAS file
(afalfa <- read(data_example("afalfa.xpt"))) # SAS transport file

# Note that where completion is available, you have a completion list of file
# format after typing read$<tab>
```

`read_write_option` *Define default read/write options and add items to it*

Description

Define the functions that `read()` or write() must call to import or export data for the different types (formats).

Usage

```
read_write_option(new_type)
```

Arguments

`new_type`	A data.frame with four columns: `type`, `read_fun`, `read_header` and `write_fun` containing each a single character string or `NA`. `type` is the usual extension for this type of file, e.g., `png` for PNG images, `read_fun`, `read_header` and `write_fun` are character strings with "pkg::fun" format ("pkg" is the package containing the function and "fun" is the function name), or just "fun" if the function is visible on the search path.

Value

The data.frame with all known formats is returned invisibly. The same data.frame is also saved in the `read_write`` option, and can be retrieved directly with` getOption("read_write")‘.

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

read(), getOption()

Examples

```
# The default options
(read_write_option())
# To add a new type:
tail(read_write_option(data.frame(type = "png", read_fun = "png::readPNG",
  read_header = NA, write_fun = "png::writePNG", comment = "PNG image")))
```

---

relative_path *Calculate path relative to a reference directory*

Description

After normalizing both `file` and `dir`, try to find a common ancestor directory to build a path for `file` relative to `dir`.

Usage

```
relative_path(file, dir = getwd())
```

Arguments

`file`	A single string with the path to a file or directory to transform as relative.
`dir`	A single string with the "reference" directory (by default, the directory provided by getwd().

Value

A single character string with the relative path, or `file` unmodified if `file` is totally unrelated to `dir`.

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

getwd(), normalizePath()

Examples

```
relative_path("/Users/me/project/file.txt", "/Users/me/project")
relative_path("/Users/me/project/subdir/file.txt", "/Users/me/project")
relative_path("/Users/me/file.txt", "/Users/me/project")
relative_path("/Users/me/subdir/file.txt", "/Users/me/project")
relative_path("/Users/file.txt", "/Users/me/project")
relative_path("/Users/subdir1/subdir2/file.txt", "/Users/me/project")
relative_path("/Unrelated/file.txt", "/Users/me/project")

relative_path("file.txt", "/Users/me/project")
relative_path("~/file.txt", "/Users/me/project")
relative_path("./file.txt", "/Users/me/project")
relative_path(file.path(getwd(), "data.io", "file.txt"))
```

---

urchin_bio *Sea urchins biometry*

Description

Various measurement on *Paracentrotus lividus* sea urchins providing from fishery (Brittany, France), or from a sea urchins farm in Normandy.

Usage

```
urchin_bio
```

Format

A data frame with 19 variables:

`origin` A **factor** with two levels: `"Culture"`, and `"Fishery"`.

`diameter1` Diameter (in mm) of the test measured at the ambitus (its widest part).

`diameter2` A second diameter (in mm) measured at the ambitus, perpendicular to the first one. The idea here is to calculate the average of `diameter1` and `diameter2` in order to eliminate the effect of possible slight departure from a nearly circular ambitus.

`height` The height of the test (in mm), measured from month to anus, thus, orthogonally to the two diameters.

`buoyant_weight` Weight (in g) of the sea urchin immersed in seawater.

`weight` Weight (in g) of the whole animal.

`solid_parts` Weight (in g) of the animal after draining its coelomic fluid out of the test.

`integuments` Weight (in g) of the sea urchin after taking out the whole content of the test (coelomic fluid, digestive tract and gonads.

`dry_integuments` Dry weight (in g) of the integuments.

`digestive_tract` Weight (in g) of the digestive tract, including its content.

`dry_digestive_tract` Dry weight (in g) of the digestive tract and its content.

`gonads` Weight (in g) of the gonads.

`dry_gonads` Dry weight (in g) of the gonads.

`skeleton` Weight of the skeleton (g), calculated as the sum of lantern + test + spines.

`lantern` Dry weight (in g) of the lantern (the jaw and teeth of the sea urchin).

`test` Dry weight (in g) of the calcareous part of the test.

`spines` Dry weight (in g) of calcareous parts of the spines.

`maturity` Gonads maturity index (integer), measured on a scale of 3 states: state 0 means the gonad is absent or spent, state 1 means it is growing but not mature, and state 2 means the gonad is mature. This should be treated as a circular variable, since the reproductive cycle is 0 -> 1 -> 2 -> 0 (spawning).

`sex` When it is possible, the sex of the animal is determined by visual inspection of the gonads (**factor** with levels `"F"` and `"M"`).

A stratified sample was performed to make sure all size classes (from 5 to 5 mm in test diameter) from each sub-population are equally represented in the dataset. Hence, the size or weight-classes distributions among each population **cannot** be studied with this dataset. However, those data are more suitable to explore allometric relationships between body measurements and/or body parts of the sea urchins over the whole size range.

For further details on the farming of these sea urchins, see here.

`urchin_growth` *Sea urchins growth*

Description

Size at age for a cohort of farmed sea urchins, *Paracentrotus lividus*.

Usage

```
urchin_growth
```

Format

An object of class `data.frame` with 7024 rows and 3 columns.

Details

The same cohort of farmed sea urchins being measured at various time intervals, the observations are not completely independent from each other: the same individuals are repeatedly measured here. As the sea urchins are not individually tagged, it is not possible to track them from one measurement to the other. However, the whole dataset is representative of the growth, and spreading of growth in a single cohort. Also, mortality could be derived from the number of measurements made at each time period, since **all** the individuals still alive are measured (no sub-sampling).

Examples

```
library(ggplot2)
ggplot(urchin_growth, aes(age, diameter)) +
  geom_jitter(alpha = 0.2) +
  xlab(label(urchin_growth$age, units = TRUE)) +
  ylab(label(urchin_growth$diameter, units = TRUE)) +
  ggtitle("Growth of a cohort of sea urchins")
```

`write` *Write data from* R *in files in different formats*

Description

Write R data into a file, in different formats.

Usage

```
write(
  data,
  file = "data",
  ncolumns = if (is.character(data)) 1 else 5,
  append = FALSE,
  sep = " ",
  type = NULL,
  fun_list = NULL,
  x,
  ...
)

## S3 method for class 'write_function_subset'
.DollarNames(x, pattern = "")
```

Arguments

`data`	An object to write in a file. The accepted class depends on what the delegated function expects (in many cases, a `data.frame` or `tibble` is just fine). If `type` is not provided, a `data.frame` is **not** suitable because only an atomic vector can be provided. Give a `matrix` instead, if you want to write tabular data, or provide `type = "txt"` for instance.
`file`	The path to the file to write to. If `type` is not provide, a connection, or a character string naming the file to write to. If `""``, print to the standard output connection. If it is "lcn
`ncolumns`	The number of columns to write the data in when `type` is provided, this is by-passed.
`append`	If `TRUE` and `type` is not provided, the `data` are appended to the connection.
`sep`	A string used to separate columns. Using `sep = "\t"` gives tab delimited output; default is `" "` when `type` is not provide, or the default provided by the delegated function if this parameter is present there.

type	The type (format) of data to read.
fun_list	The table with correspondence of the types, read, and write functions.
x	Same as data=, for compatibility with base::write(). Please, do not use both data= and x= as the same time, or an error will be generated.
...	Further arguments passed to the write function, when type is explicitly provided.
pattern	A regular expression to list matching names.

Details

This function is designed to be fully compatible with base::write(), while allowing to specify type also, and get a more interesting behavior in this case. Hence, when type is **not** provided, either with write(type = ...), or write$...(), the default code is used and a plain text file wit fields separated by spaces (be default) is written. When type is provided, then the exportation is delegated to specific functions (see data_types()) to write the data in different formats.

Value

data is returned invisibly (on the contrary to base::write() which returns NULL).

Author(s)

Philippe Grosjean phgrosjean@sciviews.org

See Also

data_types(), read(), write_csv(), base::write()

Examples

```
# Always specify type to delegate to more sophisticated functions
# (type = NULL explicitly indicated meaning: "guess from file extension")
urchin <- read("urchin_bio", package = "data.io")
write(urchin, "urchin_temporary.csv", type = NULL)
# To use a format more easily readable by Excel
write(urchin, "urchin_temporary.csv", type = "xlcsv")
# ... equivalently (and more compact)
write$xlcsv(urchin, "urchin_temporary.csv")
# Tidy up
unlink("urchin_temporary.csv")

# Write in Excel format
write$xlsx(urchin, "urchin_temporary.xlsx")
# Tidy up
unlink("urchin_temporary.xlsx")

# Use base::write() code to output atomic vectors (and matices) in text files
# when you don't specify type=
mat1 <- matrix(1:12, nrow = 4)
# To get a similar presentation in the file, you have to do:
```

```
write(t(mat1), "my_temporary_data.txt", ncolumns = 3)
file.show("my_temporary_data.txt")
# Tidy up
unlink("my_temporary_data.txt")
rm(mat1)
```

`zooplankton` *Zooplankton image analysis*

Description

Various features measured by image analysis with the package `zooimage` and `ImageJ` on samples of zooplankton originating from Tulear, Madagascar. The taxonomic classification is also provided in the `class` variable.

Usage

```
zooplankton
```

Format

A data frame with 19 variables:

`ecd` The "equivalent circular diameter", the diameter of a circle with the same area as the particle (in mm).

`area` The area of the particle on the image (in mm^2).

`perimeter` The perimeter of the particle (in mm).

`feret` The Feret diameter, that is, the largest measured diameter of the particle on the image (mm).

`major` The major axis of the ellipsoid matching the particle (mm).

`minor` The minor axis of the same ellipsoid (mm).

`mean` The mean value of the gray levels calibrated in optical density (OD), thus, unitless.

`mode` The most frequent gray level in that particle in OD.

`min` The most transparent part in OD.

`max` The most opaque part in OD.

`std_dev` The standard deviation of the OD distribution inside the particle.

`range` Transparency range as `max` - `min`.

`size` The mean diameter of the particle, as the average of `minor` and `major` (mm).

`aspect` Aspect ratio of the particle as `minor`/`major`.

`elongation` The `area` divided by the area of a circle of the same `perimeter` of the particle.

`compactness` sqrt((4/pi) * `area`) / `major`.

`transparency` 1 - (`ecd` - `size`).

`circularity` 4pi(`area` / `perimeter`^2).

`density` Density integrate by the surface covered by each gray level, i.e. O.D., inside the particle.

`class` The classification of this particle. 17 classes are made. Note that `Copepods` are `Calanoid` + `Cyclopoid` + `Harpactivoid` + `Poecilostomatoid` and they represent the most abundant zooplankton at sea.

This is a typical training set used to train a plankton classifier with machine learning algorithms. Organisms originate from various samples (different seasons, depth, etc. to take the variability into account). However, the abundance of the different classes do **not** match abundance found in each sample, i.e., rare classes are over-represented in this training set. Only zooplankton classes are present in the dataset. Full data also contains classes for phytoplankton, marine snow, etc. Take care that several variables are correlated!

Source

Grosjean, Ph & K. Denis (2004). Supervised classification of images, applied to plankton samples using R and ZooImage. Chap.12 of Data Mining Applications with R. Zhao, Y. & Y. Cen (eds). Elsevier. Pp 331-365. https://doi.org/10.1016/C2012-0-00333-X.

Examples

```
table(zooplankton$class)
library(ggplot2)
ggplot(zooplankton, aes(circularity, transparency, color = class)) +
  geom_point()
```

Index