Package 'rio'

June 18, 2017

Type Package

Title A Swiss-Army Knife for Data I/O

Version 0.5.5

Date 2017-06-17

Description Streamlined data import and export by making assumptions that the user is probably willing to make: 'import()' and 'export()' determine the data structure from the file extension, reasonable defaults are used for data import and export (e.g., 'stringsAsFactors=FALSE'), web-based import is natively supported (including from SSL/HTTPS), compressed files can be read directly without explicit decompression, and fast import packages are used where appropriate. An additional convenience function, 'convert()', provides a simple method for converting between file types.

URL https://github.com/leeper/rio

BugReports https://github.com/leeper/rio/issues

Depends R (>= 2.15.0)

Imports tools, stats, utils, foreign, haven (> 0.2.1), curl (>= 0.6), data.table (>= 1.9.8), readxl (>= 0.1.1), openxlsx, tibble

Suggests datasets, bit64, testthat, knitr, magrittr, clipr, csvy, feather, fst, jsonlite, readODS (>= 1.6.4), readr, rmatio, xml2 (>= 1.0.0), yaml

License GPL-2

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Jason Becker [ctb],
Chung-hong Chan [aut],
Geoffrey CH Chan [ctb],
Thomas J. Leeper [aut, cre],
Christopher Gandrud [ctb],
Andrew MacDonald [ctb],
Ista Zahn [ctb],
Stanislaus Stadlmann [ctb]

Maintainer Thomas J. Leeper <thosjleeper@gmail.com>

Repository CRAN

Date/Publication 2017-06-18 13:20:01 UTC

R topics documented:

`.import` *rio Extensions*

Description

Writing Import/Export Extensions for rio

Usage

```
.import(file, ...)

## Default S3 method:
.import(file, ...)

.export(file, x, ...)

## Default S3 method:
.export(file, x, ...)
```

Arguments

`file`	A character string naming a file.
`...`	Additional arguments passed to methods.
`x`	A data frame or matrix to be written into a file.

Details

rio implements format-specific S3 methods for each type of file that can be imported from or exported to. This happens via internal S3 generics, `.import` and `.export`. It is possible to write new methods like with any S3 generic (e.g., `print`).

As an example, `.import.rio_csv` imports from a comma-separated values file. If you want to produce a method for a new filetype with extension "myfile", you simply have to create a function called `.import.rio_myfile` that implements a format-specific importing routine and returns a data.frame. rio will automatically recognize new S3 methods, so that you can then import your file using: `import("file.myfile")`.

As general guidance, if an import method creates many attributes, these attributes should be stored — to the extent possible — in variable-level attributes fields. These can be "gathered" to the data.frame level by the user via `gather_attrs`.

Value

For `.import`, an R data.frame. For `.export`, `file`, invisibly.

See Also

`import`, `export`

---

characterize *Character conversion of labelled data*

Description

Convert labelled variables to character or factor

Usage

```
characterize(x, ...)

factorize(x, ...)

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

## S3 method for class 'data.frame'
characterize(x, ...)

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

## S3 method for class 'data.frame'
factorize(x, ...)
```

Arguments

x	A vector or data frame.
...	additional arguments passed to methods

Details

`characterize` converts a vector with a `labels` attribute of named levels into a character vector. `factorize` does the same but to factors. This can be useful at two stages of a data workflow: (1) importing labelled data from metadata-rich file formats (e.g., Stata or SPSS), and (2) exporting such data to plain text files (e.g., CSV) in a way that preserves information.

See Also

`gather_attrs`

Examples

```
# vector method
x <- structure(1:4, labels = c("A" = 1, "B" = 2, "C" = 3))
characterize(x)
factorize(x)

# data frame method
x <- data.frame(v1 = structure(1:4, labels = c("A" = 1, "B" = 2, "C" = 3)),
                v2 = structure(c(1,0,0,1), labels = c("foo" = 0, "bar" = 1)))
str(factorize(x))
str(characterize(x))

# comparison of exported file contents
import(export(x, "example.csv"))
import(export(factorize(x), "example.csv"))

# cleanup
unlink("example.csv")
```

---

convert *Convert from one file format to another*

Description

This function constructs a data frame from a data file using `import` and uses `export` to write the data to disk in the format indicated by the file extension.

Usage

```
convert(in_file, out_file, in_opts = list(), out_opts = list())
```

Arguments

in_file	A character string naming an input file.
out_file	A character string naming an output file.
in_opts	A named list of options to be passed to import.
out_opts	A named list of options to be passed to export.

Value

A character string containing the name of the output file (invisibly).

See Also

Luca Braglia has created a Shiny app called rioweb that provides access to the file conversion features of rio through a web browser. The app is featured in the RStudio Shiny Gallery.

Examples

```
# create a file to convert
export(mtcars, "mtcars.dta")

# convert Stata to CSV and open converted file
convert("mtcars.dta", "mtcars.csv")
head(import("mtcars.csv"))

# correct an erroneous file format
export(mtcars, "mtcars.csv", format = "tsv")
convert("mtcars.csv", "mtcars.csv", in_opts = list(format = "tsv"))

# convert serialized R data.frame to JSON
export(mtcars, "mtcars.rds")
convert("mtcars.rds", "mtcars.json")

# cleanup
unlink("mtcars.csv")
unlink("mtcars.dta")
unlink("mtcars.rds")
unlink("mtcars.json")

## Not run:
# convert from the command line:
Rscript -e "rio::convert('mtcars.dta', 'mtcars.csv')"

## End(Not run)
```

`export` *Export*

Description

Write data.frame to a file

Usage

```
export(x, file, format, ...)
```

Arguments

`x`	A data frame or matrix to be written into a file. (An exception to this is that `x` can be a list of data frames if the output file format is an Excel .xlsx workbook, .Rdata file, or HTML file. See examples.)
`file`	A character string naming a file. Must specify `file` and/or `format`.
`format`	An optional character string containing the file format, which can be used to override the format inferred from `file` or, in lieu of specifying `file`, a file with the symbol name of `x` and the specified file extension will be created. Must specify `file` and/or `format`. Shortcuts include: "," (for comma-separated values), ";" (for semicolon-separated values), "|" (for pipe-separated values), and "dump" for `dump`.
`...`	Additional arguments for the underlying export functions. See examples.

Details

This function exports a data frame or matrix into a file with file format based on the file extension (or the manually specified format, if `format` is specified).

The output file can be to a compressed directory, simply by adding an appropriate additional extensiont to the `file` argument, such as: "mtcars.csv.tar", "mtcars.csv.zip", or "mtcars.csv.gz".

`export` supports many file formats. See the documentation for the underlying export functions for optional arguments that can be passed via `...`

- Comma-separated data (.csv), using `fwrite` or, if `fwrite = TRUE`, `write.table` with `row.names = FALSE`.
- Pipe-separated data (.psv), using `fwrite` or, if `fwrite = TRUE`, `write.table` with `sep = '|'` and `row.names = FALSE`.
- Tab-separated data (.tsv), using `fwrite` or, if `fwrite = TRUE`, `write.table` with `row.names = FALSE`.
- SAS (.sas7bdat), using `write_sas`.
- SPSS (.sav), using `write_sav`
- Stata (.dta), using `write_dta`. Note that variable/column names containing dots (.) are not allowed and will produce an error.

- Excel (.xlsx), using write.xlsx. Use which to specify a sheet name and overwrite to decide whether to overwrite an existing file or worksheet (the default) or add the data as a new worksheet (with overwrite = FALSE). x can also be a list of data frames; the list entry names are used as sheet names.
- R syntax object (.R), using dput (by default) or dump (if format = 'dump')
- Saved R objects (.RData,.rda), using save. In this case, x can be a data frame, a named list of objects, an R environment, or a character vector containing the names of objects if a corresponding envir argument is specified.
- Serialized R objects (.rds), using saveRDS
- "XBASE" database files (.dbf), using write.dbf
- Weka Attribute-Relation File Format (.arff), using write.arff
- Fixed-width format data (.fwf), using write.table with row.names = FALSE, quote = FALSE, and col.names = FALSE
- gzip comma-separated data (.csv.gz), using write.table with row.names = FALSE
- CSVY (CSV with a YAML metadata header) using write_csvy. The YAML header lines are preceded by R comment symbols (#) by default; this can be turned off by passing a comment_header = FALSE argument to export. Setting fwrite = TRUE (the default) will rely on fwrite for much faster export.
- Feather R/Python interchange format (.feather), using write_feather
- Fast storage (.fst), using write.fst
- JSON (.json), using toJSON
- Matlab (.mat), using write.mat
- OpenDocument Spreadsheet (.ods), using write_ods. (Currently only single-sheet exports are supported.)
- HTML (.html), using a custom method based on xml_add_child to create a simple HTML table and write_xml to write to disk.
- XML (.xml), using a custom method based on xml_add_child to create a simple XML tree and write_xml to write to disk.
- YAML (.yml), using as.yaml
- Clipboard export (on Windows and Mac OS), using write.table with row.names = FALSE

When exporting a data set that contains label attributes (e.g., if imported from an SPSS or Stata file) to a plain text file, characterize can be a useful pre-processing step that records value labels into the resulting file (e.g., export(characterize(x), "file.csv")) rather than the numeric values.

Value

The name of the output file as a character string (invisibly).

See Also

.export, characterize, import, convert

Examples

```
library("datasets")
# specify only `file` argument
export(mtcars, "mtcars.csv")

## Not run:
# Stata does not recognize variables names with '.'
export(mtcars, "mtcars.dta")

## End(Not run)

# specify only `format` argument
"mtcars.dta" %in% dir()
export(mtcars, format = "stata")
"mtcars.dta" %in% dir()

# specify `file` and `format` to override default format
export(mtcars, file = "mtcars.txt", format = "csv")

# export multiple objects to Rdata
export(list(mtcars = mtcars, iris = iris), "mtcars.rdata")
export(c("mtcars", "iris"), "mtcars.rdata")

# export to JSON
export(mtcars, "mtcars.json")

# pass arguments to underlying export function
export(mtcars, "mtcars.csv", col.names = FALSE)

# write data to .R syntax file and append additional data
export(mtcars, file = "data.R", format = "dump")
export(mtcars, file = "data.R", format = "dump", append = TRUE)
source("data.R", echo = TRUE)

# write to an Excel workbook
## Not run:
  ## export a single data frame
  export(mtcars, "mtcars.xlsx")

  ## export a list of data frames as worksheets
  export(list(a = mtcars, b = iris), "multisheet.xlsx")

  ## export, adding sheet to an existing workbook
  export(iris, "mtcars.xlsx", which = "iris", overwrite = FALSE)

## End(Not run)

# write data to a zip-compressed CSV
export(mtcars, "mtcars.csv.zip")

# cleanup
unlink("mtcars.csv")
```

```
unlink("mtcars.dta")
unlink("mtcars.json")
unlink("mtcars.rdata")
unlink("data.R")
unlink("mtcars.csv.zip")
```

---

gather_attrs *Gather attributes from data frame variables*

Description

gather_attrs moves variable-level attributes to the data frame level and spread_attrs reverses that operation.

Usage

```
gather_attrs(x)

spread_attrs(x)
```

Arguments

x	A data frame.

Details

import attempts to standardize the return value from the various import functions to the extent possible, thus providing a uniform data structure regardless of what import package or function is used. It achieves this by storing any optional variable-related attributes at the variable level (i.e., an attribute for mtcars$mpg is stored in attributes(mtcars$mpg) rather than attributes(mtcars)). gather_attrs moves these to the data frame level (i.e., in attributes(mtcars)). spread_attrs moves attributes back to the variable level.

Value

x, with variable-level attributes stored at the data frame level.

See Also

import, characterize

Examples

```
e <- import("http://www.stata-press.com/data/r13/auto.dta")
str(e)
g <- gather_attrs(e)
str(attributes(e))
str(g)
```

`import` *Import*

Description

Read in a data.frame from a file

Usage

```
import(file, format, setclass, which, ...)
```

Arguments

`file`	A character string naming a file, URL, or single-file .zip or .tar archive.
`format`	An optional character string code of file format, which can be used to override the format inferred from `file`. Shortcuts include: "," (for comma-separated values), ";" (for semicolon-separated values), and "|" (for pipe-separated values).
`setclass`	An optional character vector specifying one or more classes to set on the import. By default, all the return object is always a "data.frame". Allowed values for this might be "tbl_df", "tbl", or "tibble" (if using dplyr) or "data.table" (if using data.table). Other values are ignored such that a data.frame is returned.
`which`	This argument is used to control import from multi-object files; as a rule `import` only ever returns a single data frame. (Use `import_list` to import multiple data frames from a multi-object file.) If `file` is a compressed directory, `which` can be either a character string specifying a filename or an integer specifying which file (in locale sort order) to extract from the compressed directory. For Excel spreadsheets, this can be used to specify a sheet number. For .Rdata files, this can be an object name. For HTML files, which table to exract (from document order). Ignored otherwise. A character string value will be used as a regular expression, such that the extracted file is the first match of the regular expression against the file names in the archive.
`...`	Additional arguments passed to the underlying import functions. For example, this can control column classes for delimited file types, or control the use of haven for Stata and SPSS or readxl for Excel (.xlsx) format. See details below.

Details

This function imports a data frame or matrix from a data file with the file format based on the file extension (or the manually specified format, if `format` is specified).

`import` supports the following file formats:

- Comma-separated data (.csv), using `fread` or, if `fread = FALSE`, `read.table` with `row.names = FALSE` and `stringsAsFactors = FALSE`
- Pipe-separated data (.psv), using `fread` or, if `fread = FALSE`, `read.table` with `sep = '|'`, `row.names = FALSE` and `stringsAsFactors = FALSE`

- Tab-separated data (.tsv), using `fread` or, if `fread = FALSE`, `read.table` with `row.names = FALSE` and `stringsAsFactors = FALSE`
- SAS (.sas7bdat), using `read_sas`.
- SPSS (.sav), using `read_sav`. If `haven = FALSE`, `read.spss` can be used.
- Stata (.dta), using `read_dta`. If `haven = FALSE`, `read.dta` can be used.
- SAS XPORT (.xpt), using `read.xport`.
- SPSS Portable Files (.por), using `read_por`.
- Excel (.xls and .xlsx), using `read_excel`. Use `which` to specify a sheet number. For .xlsx files, it is possible to set `readxl = FALSE`, so that `read.xlsx` can be used instead of readxl (the default).
- R syntax object (.R), using `dget`
- Saved R objects (.RData,.rda), using `load` for single-object .Rdata files. Use `which` to specify an object name for multi-object .Rdata files.
- Serialized R objects (.rds), using `readRDS`
- Epiinfo (.rec), using `read.epiinfo`
- Minitab (.mtp), using `read.mtp`
- Systat (.syd), using `read.systat`
- "XBASE" database files (.dbf), using `read.dbf`
- Weka Attribute-Relation File Format (.arff), using `read.arff`
- Data Interchange Format (.dif), using `read.DIF`
- Fortran data (no recognized extension), using `read.fortran`
- Fixed-width format data (.fwf), using a faster version of `read.fwf` that requires a `widths` argument and by default in rio has `stringsAsFactors = FALSE`. If `readr = TRUE`, import will be performed using `read_fwf`, where `widths` should be: `NULL`, a vector of column widths, or the output of `fwf_empty`, `fwf_widths`, or `fwf_positions`.
- gzip comma-separated data (.csv.gz), using `read.table` with `row.names = FALSE` and `stringsAsFactors = FALSE`
- CSVY (CSV with a YAML metadata header) using `read_csvy`.
- Feather R/Python interchange format (.feather), using `read_feather`
- Fast storage (.fst), using `read.fst`
- JSON (.json), using `fromJSON`
- Matlab (.mat), using `read.mat`
- OpenDocument Spreadsheet (.ods), using `read_ods`. Use `which` to specify a sheet number.
- Single-table HTML documents (.html), using `read_html`. The data structure will only be read correctly if the HTML file can be converted to a list via `as_list`.
- Shallow XML documents (.xml), using `read_xml`. The data structure will only be read correctly if the XML file can be converted to a list via `as_list`.
- YAML (.yml), using `yaml.load`
- Clipboard import (on Windows and Mac OS), using `read.table` with `row.names = FALSE`
- Google Sheets, as Comma-separated data (.csv)

`import` attempts to standardize the return value from the various import functions to the extent possible, thus providing a uniform data structure regardless of what import package or function is used. It achieves this by storing any optional variable-related attributes at the variable level (i.e., an attribute for `mtcars$mpg` is stored in `attributes(mtcars$mpg)` rather than `attributes(mtcars)`). If you would prefer these attributes to be stored at the data.frame-level (i.e., in `attributes(mtcars)`), see `gather_attrs`.

After importing metadata-rich file formats (e.g., from Stata or SPSS), it may be helpful to recode labelled variables to character or factor using `characterize` or `factorize` respectively.

Value

A data frame. If `setclass` is used, this data frame may have additional class attribute values, such as "tibble" or "data.table".

Note

For csv and txt files with row names exported from `export`, it may be helpful to specify `row.names` as the column of the table which contain row names. See example below.

See Also

`import_list`, `.import`, `characterize`, `gather_attrs`, `export`, `convert`

Examples

```
# create CSV to import
export(iris, "iris1.csv")

# specify `format` to override default format
export(iris, "iris.tsv", format = "csv")
stopifnot(identical(import("iris1.csv"), import("iris.tsv", format = "csv")))

# import CSV as a `data.table`
stopifnot(inherits(import("iris1.csv", setclass = "data.table"), "data.table"))
stopifnot(inherits(import("iris1.csv", setclass = "data.table"), "data.table"))

# pass arguments to underlying import function
iris1 <- import("iris1.csv")
identical(names(iris), names(iris1))

export(iris, "iris2.csv", col.names = FALSE)
iris2 <- import("iris2.csv")
identical(names(iris), names(iris2))

# set class for the response data.frame as "tbl_df" (from dplyr)
stopifnot(inherits(import("iris1.csv", setclass = "tbl_df"), "tbl_df"))

# cleanup
unlink("iris.tsv")
unlink("iris1.csv")
unlink("iris2.csv")
```

`import_list` *Import list of data frames*

Description

Use `import` to import a list of data frames from a vector of file names or from a multi-object file (Excel workbook, .Rdata file, zip directory, or HTML file)

Usage

```
import_list(file, setclass, which, rbind = FALSE, rbind_label = "_file",
  rbind_fill = TRUE, ...)
```

Arguments

`file`	A character string containing a single file name for a multi-object file (e.g., Excel workbook, zip directory, or HTML file), or a vector of file paths for multiple files to be imported.
`setclass`	An optional character vector specifying one or more classes to set on the import. By default, all the return object is always a "data.frame". Allowed values for this might be "tbl_df", "tbl", or "tibble" (if using dplyr) or "data.table" (if using data.table). Other values are ignored such that a data.frame is returned.
`which`	If `file` is a single file path, this specifies which objects should be extracted (passed to `import`'s `which` argument). Ignored otherwise.
`rbind`	A logical indicating whether to pass the import list of data frames through `rbindlist`.
`rbind_label`	If `rbind = TRUE`, a character string specifying the name of a column to add to the data frame indicating its source file.
`rbind_fill`	If `rbind = TRUE`, a logical indicating whether to set the `fill = TRUE` (and fill missing columns with `NA`).
`...`	Additional arguments passed to `import`. Behavior may be unexpected if files are of different formats.

Value

If `rbind=FALSE` (the default), a list of a data frames. Otherwise, that list is passed to `rbindlist` with `fill = TRUE` and returns a data frame object of class set by the `setclass` argument; if this operation fails, the list is returned.

See Also

`import`

Examples

```
library('datasets')
export(list(mtcars1 = mtcars[1:10,],
            mtcars2 = mtcars[11:20,],
            mtcars2 = mtcars[21:32,]), "mtcars.xlsx")

# import a single file from multi-object workbook
str(import("mtcars.xlsx", which = "mtcars1"))

# import all worksheets
str(import_list("mtcars.xlsx"), 1)

# import and rbind all worksheets
mtcars2 <- import_list("mtcars.xlsx", rbind = TRUE)
all.equal(mtcars2, mtcars, check.attributes = FALSE)

# import multiple files
export(mtcars, "mtcars.csv")
export(mtcars, "iris.csv")
str(import_list(dir(pattern = "csv$")), 1)

# cleanup
unlink("mtcars.xlsx")
unlink("mtcars.csv")
unlink("iris.csv")
```

---

install_formats *Install rio's 'Suggests' Dependencies*

Description

This function installs various 'Suggests' dependencies for rio that expand its support to the full range of support import and export formats. These packages are not installed or loaded by default in order to create a slimmer and faster package build, install, and load.

Usage

```
install_formats(...)
```

Arguments

...	Additional arguments passed to `install.packages`.

Value

NULL

rio *A Swiss-Army Knife for Data I/O*

Description

The aim of rio is to make data file input and output as easy as possible. `export` and `import` serve as a Swiss-army knife for painless data I/O for data from almost any file format by inferring the data structure from the file extension, natively reading web-based data sources, setting reasonable defaults for import and export, and relying on efficient data import and export packages. An additional convenience function, `convert`, provides a simple method for converting between file types.

Note that some of rio's functionality is provided by 'Suggests' dependendencies, meaning they are not installed by default. Use `install_formats` to make sure these packages are available for use.

References

GREA provides an RStudio add-in to import data using rio.

See Also

`import`, `import_list`, `export`, `convert`, `install_formats`

Examples

```
# export
library("datasets")
export(mtcars, "mtcars.csv") # comma-separated values
export(mtcars, "mtcars.rds") # R serialized
export(mtcars, "mtcars.sav") # SPSS

# import
x <- import("mtcars.csv")
y <- import("mtcars.rds")
z <- import("mtcars.sav")

# convert
convert("mtcars.sav", "mtcars.dta")

# cleanup
unlink(c("mtcars.csv", "mtcars.rds", "mtcars.sav", "mtcars.dta"))
```

Index