Package ‘rio’

November 22, 2021

Type Package

Title A Swiss-Army Knife for Data I/O

Version 0.5.29

Date 2021-11-08

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 (>= 1.1.2), curl (>= 0.6),
data.table (>= 1.9.8), readxl (>= 0.1.1), openxlsx, tibble

Suggests datasets, bit64, testthat, knitr, magrittr, arrow, clipr,
feather, fst, hexView, jsonlite, pzfx, readODS (>= 1.6.4),
readr, rmarkdown, rmatio, xml2 (>= 1.2.0), yaml

License GPL-2

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author Jason Becker [ctb],
Chung-hong Chan [aut] (<https://orcid.org/0000-0002-6232-7530>),
Geoffrey CH Chan [ctb],
Thomas J. Leeper [aut, cre] (<https://orcid.org/0000-0003-4097-6326>),
Christopher Gandrud [ctb],
Andrew MacDonald [ctb],

Ista Zahn [ctb],
Stanislaus Stadlmann [ctb],
Ruaridh Williamson [ctb],
Patrick Kennedy [ctb],
Ryan Price [ctb],
Trevor L Davis [ctb],
Nathan Day [ctb],
Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>),
Alex Bokov [ctb] (<https://orcid.org/0000-0002-0511-9815>)

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

Repository CRAN

Date/Publication 2021-11-22 07:40:02 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`

`arg_reconcile` *Reconcile an argument list to any function signature.*

Description

Adapt an argument list to a function excluding arguments that will not be recognized by it, redundant arguments, and un-named arguments.

Usage

```
arg_reconcile(
  fun,
  ...,
  .args = alist(),
  .docall = FALSE,
  .include = c(),
  .exclude = c(),
  .remap = list(),
  .warn = TRUE,
  .error = "default",
  .finish = identity
)
```

Arguments

`fun`	A function to which an argument list needs to be adapted. Use the unquoted name of the function. If it's in a different package then the fully qualified unquoted name (e.g. `utils::read.table`)
`...`	An arbitrary list of named arguments (unnamed ones will be ignored). Arguments in `.args` are overridden by arguments of the same name (if any) in `...`
`.args`	A list or `alist` of named arguments, to be merged with `...`. Arguments in `.args` are overridden by arguments of the same name (if any) in `...`
`.docall`	If set to `TRUE` will not only clean up the arguments but also execute `fun` with those arguments (`FALSE` by default) and return the results
`.include`	Whitelist. If not empty, only arguments named here will be permitted, and only if they satisfy the conditions implied by the other arguments. Evaluated before `.remap`.
`.exclude`	Blacklist. If not empty, arguments named here will be removed even if they satisfy the conditions implied by the other arguments. Evaluated before `.remap`.
`.remap`	An optional named character vector or named list of character values for standardizing arguments that play the same role but have different names in different functions. Evaluated after `.exclude` and `.include`.
`.warn`	Whether to issue a warning message (default) when invalid arguments need to be discarded.
`.error`	If specified, should be the object to return in the event of error. This object will have the error as its `error` attribute. If not specified an ordinary error is thrown with an added hint on the documentation to read for troubleshooting. Ignored if `.docall` is `FALSE`. The point of doing this is fault-tolerance– if this function is part of a lengthy process where you want to document an error but keep going, you can set `.error` to some object of a compatible type. That object will be returned in the event of error and will have as its `"error"` attribute the error object.
`.finish`	A function to run on the result before returning it. Ignored if `.docall` is `FALSE`.

Value

Either a named list or the result of calling `fun` with the supplied arguments

---

`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, coerce_character = FALSE, ...)

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

Arguments

`x`	A vector or data frame.
`...`	additional arguments passed to methods
`coerce_character`	A logical indicating whether to additionally coerce character columns to factor (in `factorize`). Default `FALSE`.

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, csv_file <- tempfile(fileext = ".csv")))
import(export(factorize(x), csv_file))

# cleanup
unlink(csv_file)
```

---

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.

Examples

```
# create a file to convert
export(mtcars, dta_file <- tempfile(fileext = ".dta"))

# convert Stata to CSV and open converted file
convert(dta_file, csv_file <- tempfile(fileext = ".csv"))
head(import(csv_file))

# correct an erroneous file format
export(mtcars, csv_file2 <- tempfile(fileext = ".csv"), format = "tsv")
convert(csv_file2, csv_file, in_opts = list(format = "tsv"))

# convert serialized R data.frame to JSON
export(mtcars, rds_file <- tempfile(fileext = ".rds"))
convert(rds_file, json_file <- tempfile(fileext = ".json"))

# cleanup
unlink(csv_file)
unlink(csv_file2)
unlink(rds_file)
unlink(dta_file)
unlink(json_file)

## Not run: \donttest{
# 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. Exceptions to this rule are that `x` can be a list of data frames if the output file format is an Excel .xlsx workbook, .Rdata file, or HTML file, or a variety of R objects if the output file format is RDS or JSON. See examples.) To export a list of data frames to multiple files, use `export_list` instead.
`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. This can be used to specify non-standard arguments. 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`.
- SAS XPORT (.xpt), using `write_xpt`.
- SPSS (.sav), using `write_sav`
- SPSS compressed (.zsav), 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`. Existing workbooks are overwritten unless `which` is specified, in which case only the specified sheet (if it exists) is overwritten. If the file exists but the `which` sheet does not, data are added as a new sheet to the existing workbook. `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`. In this case, `x` can be any serializable R object.
- "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 `fwrite`.
- Apache Arrow Parquet (.parquet), using `write_parquet`
- Feather R/Python interchange format (.feather), using `write_feather`
- Fast storage (.fst), using `write.fst`
- JSON (.json), using `toJSON`. In this case, `x` can be a variety of R objects, based on class mapping conventions in this paper: https://arxiv.org/abs/1403.2805.
- 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.

Use `export_list` to export a list of dataframes to separate files.

Value

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

See Also

`.export`, `characterize`, `import`, `convert`, `export_list`

Examples

```
library("datasets")
# specify only `file` argument
export(mtcars, f1 <- tempfile(fileext = ".csv"))

## Not run:
wd <- getwd()
setwd(tempdir())
# Stata does not recognize variables names with '.'
export(mtcars, f2 <- tempfile(fileext = ".dta"))

# specify only `format` argument
f2 %in% tempdir()
export(mtcars, format = "stata")
"mtcars.dta" %in% dir()
```

```
setwd(wd)

## End(Not run)
# specify `file` and `format` to override default format
export(mtcars, file = f3 <- tempfile(fileext = ".txt"), format = "csv")

# export multiple objects to Rdata
export(list(mtcars = mtcars, iris = iris), f4 <- tempfile(fileext = ".rdata"))
export(c("mtcars", "iris"), f4)

# export to non-data frame R object to RDS or JSON
export(mtcars$cyl, f5 <- tempfile(fileext = ".rds"))
export(list(iris, mtcars), f6 <- tempfile(fileext = ".json"))

# pass arguments to underlying export function
export(mtcars, f7 <- tempfile(fileext = ".csv"), col.names = FALSE)

# write data to .R syntax file and append additional data
export(mtcars, file = f8 <- tempfile(fileext = ".R"), format = "dump")
export(mtcars, file = f8, format = "dump", append = TRUE)
source(f8, echo = TRUE)

# write to an Excel workbook
## Not run:
  ## export a single data frame
  export(mtcars, f9 <- tempfile(fileext = ".xlsx"))

  ## export NAs to Excel as missing via args passed to `...`
  mtcars$drat <- NA_real_
  mtcars %>% export(f10 <- tempfile(fileext = ".xlsx"), keepNA = TRUE)

  ## export a list of data frames as worksheets
  export(list(a = mtcars, b = iris), f11 <- tempfile(fileext = ".xlsx"))

  ## export, adding a new sheet to an existing workbook
  export(iris, f12 <- tempfile(fileext = ".xlsx"), which = "iris")

## End(Not run)

# write data to a zip-compressed CSV
export(mtcars, f13 <- tempfile(fileext = ".csv.zip"))

# cleanup
unlink(f1)
# unlink(f2)
unlink(f3)
unlink(f4)
unlink(f5)
unlink(f6)
unlink(f7)
unlink(f8)
# unlink(f9)
# unlink(f10)
# unlink(f11)
# unlink(f12)
# unlink(f13)
```

`export_list` *Export list of data frames to files*

Description

Use `export` to export a list of data frames to a vector of file names or a filename pattern.

Usage

```
export_list(x, file, ...)
```

Arguments

`x`	A list of data frames to be written to files.
`file`	A character vector string containing a single file name with a `%s` wildcard placeholder, or a vector of file paths for multiple files to be imported. If `x` elements are named, these will be used in place of `%s`, otherwise numbers will be used; all elements must be named for names to be used.
`...`	Additional arguments passed to `export`.

Details

`export` can export a list of data frames to a single multi-dataset file (e.g., an Rdata or Excel .xlsx file). Use `export_list` to export such a list to *multiple* files.

Value

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

See Also

`import`, `import_list`, `export`

Examples

```
library('datasets')
export(list(mtcars1 = mtcars[1:10,],
            mtcars2 = mtcars[11:20,],
            mtcars3 = mtcars[21:32,]),
    xlsx_file <- tempfile(fileext = ".xlsx")
)

# import all worksheets
mylist <- import_list(xlsx_file)
```

```
# re-export as separate named files
csv_files1 <- sapply(1:3, function(x) tempfile(fileext = paste0("-", x, ".csv")))
export_list(mylist, file = csv_files1)

# re-export as separate files using a name pattern
export_list(mylist, file = csv_files2 <- tempfile(fileext = "%s.csv"))

# cleanup
unlink(xlsx_file)
unlink(csv_files1)
unlink(csv_files2)
```

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 <- try(import("http://www.stata-press.com/data/r13/auto.dta"))
if (!inherits(e, "try-error")) {
  str(e)
  g <- gather_attrs(e)
  str(attributes(e))
  str(g)
}
```

`get_ext` *Get File Type from Extension*

Description

A utility function to retrieve the file type from a file extension (via its filename/path/URL)

Usage

```
get_ext(file)
```

Arguments

`file`	A character string containing a filename, file path, or URL.

Value

A characters string containing a file type recognized by rio.

`import` *Import*

Description

Read in a data.frame from a file. Exceptions to this rule are Rdata, RDS, and JSON input file formats, which return the originally saved object without changing its class.

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, the return object is always a “data.frame”. Allowed values include “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 name or number. For .Rdata files, this can be an object name. For HTML files, it identifies which table to extract (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`.
- SAS XPORT (.xpt), using `read_xpt` or, if `haven = FALSE`, `read.xport`.
- SPSS (.sav), using `read_sav`. If `haven = FALSE`, `read.spss` can be used.
- SPSS compressed (.zsav), using `read_sav`.
- Stata (.dta), using `read_dta`. If `haven = FALSE`, `read.dta` can be used.
- 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. This can be any R object (not just a data frame).
- Serialized R objects (.rds), using `readRDS`. This can be any R object (not just a data frame).
- 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 `fread`.
- Apache Arrow Parquet (.parquet), using `read_parquet`
- Feather R/Python interchange format (.feather), using `read_feather`
- Fast storage (.fst), using `read.fst`
- JSON (.json), using `fromJSON`
- Matlab (.mat), using `read.mat`
- EViews (.wf1), using `readEViews`
- 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)
- GraphPad Prism (.pzfx) using `read_pzfx`

`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, csv_file <- tempfile(fileext = ".csv"))

# specify `format` to override default format
export(iris, tsv_file <- tempfile(fileext = ".tsv"), format = "csv")
stopifnot(identical(import(csv_file), import(tsv_file, format = "csv")))

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

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

export(iris, csv_file2 <- tempfile(fileext = ".csv"), col.names = FALSE)
iris2 <- import(csv_file2)
identical(names(iris), names(iris2))

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

# non-data frame formats supported for RDS, Rdata, and JSON
export(list(mtcars, iris), rds_file <- tempfile(fileext = ".rds"))
li <- import(rds_file)
identical(names(mtcars), names(li[[1]]))

# cleanup
unlink(csv_file)
unlink(csv_file2)
unlink(tsv_file)
unlink(rds_file)
```

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, the return object is always a "data.frame". Allowed values include "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, export_list, export

Examples

```
library('datasets')
export(list(mtcars1 = mtcars[1:10,],
            mtcars2 = mtcars[11:20,],
            mtcars3 = mtcars[21:32,]),
    xlsx_file <- tempfile(fileext = ".xlsx")
)

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

# import all worksheets
str(import_list(xlsx_file), 1)

# import and rbind all worksheets
mtcars2 <- import_list(xlsx_file, rbind = TRUE)
all.equal(mtcars2[,-12], mtcars, check.attributes = FALSE)

# import multiple files
wd <- getwd()
setwd(tempdir())
export(mtcars, "mtcars1.csv")
export(mtcars, "mtcars2.csv")
str(import_list(dir(pattern = "csv$")), 1)
unlink(c("mtcars1.csv", "mtcars2.csv"))
setwd(wd)

# cleanup
unlink(xlsx_file)
```

`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`

`is_file_text` *Determine whether a file is "plain-text" or some sort of binary format*

Description

Determine whether a file is "plain-text" or some sort of binary format

Usage

```
is_file_text(file, maxsize = Inf, text_bytes = as.raw(c(7:16, 18, 19, 32:255)))
```

Arguments

`file`	Path to the file
`maxsize`	Maximum number of bytes to read
`text_bytes`	Which characters are used by normal text (though not necessarily just ASCII). To detect just ASCII, the following value can be used: `as.raw(c(7:16,18,19,32:127))`

Value

A logical

Examples

```
library(datasets)
export(iris, yml_file <- tempfile(fileext = ".yml"))
is_file_text(yml_file) # TRUE

export(iris, sav_file <- tempfile(fileext = ".sav"))
is_file_text(sav_file) # FALSE

# cleanup
unlink(yml_file)
unlink(sav_file)
```

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, export_list, convert, install_formats

Examples

```
# export
library("datasets")
export(mtcars, csv_file <- tempfile(fileext = ".csv")) # comma-separated values
export(mtcars, rds_file <- tempfile(fileext = ".rds")) # R serialized
export(mtcars, sav_file <- tempfile(fileext = ".sav")) # SPSS

# import
x <- import(csv_file)
y <- import(rds_file)
z <- import(sav_file)

# convert sav (SPSS) to dta (Stata)
convert(sav_file, dta_file <- tempfile(fileext = ".dta"))

# cleanup
unlink(c(csv_file, rds_file, sav_file, dta_file))
```

Index