Package: rio (via r-universe)

October 25, 2024

Type Package

Title A Swiss-Army Knife for Data I/O

Version 1.2.3

Description Streamlined data import and export by making assumptions that the user is probably willing to make: 'import()' and 'export()' determine the data format from the file extension, reasonable defaults are used for data import and export, web-based import is natively supported (including from SSL/HTTPS), compressed files can be read directly, and fast import packages are used where appropriate. An additional convenience function, 'convert()', provides a simple method for converting between file types.

URL https://gesistsa.github.io/rio/, https://github.com/gesistsa/rio

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

Depends R (>= 4.0)

Imports tools, stats, utils, foreign, haven (>= 1.1.2), curl (>= 0.6), data.table (>= 1.11.2), readxl (>= 0.1.1), tibble, writexl, lifecycle, R.utils, readr

Suggests datasets, bit64, testthat, knitr, magrittr, clipr, fst, hexView, jsonlite, pzfx, readODS (>= 2.1.0), rmarkdown, rmatio, xml2 (>= 1.2.0), yaml, qs, arrow (>= 0.17.0), stringi, withr, nanoparquet

License GPL-2

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Config/Needs/website gesistsa/tsatemplate

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

RemoteUrl https://github.com/gesistsa/rio

RemoteRef HEAD

RemoteSha 727d50f95dc89027ce5f1ac49f41426725de37bc

Contents

`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.

Value

a character vector (for characterize) or factor vector (for factorize)

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))

## Application
csv_file <- tempfile(fileext = ".csv")
## comparison of exported file contents
import(export(x, csv_file))
import(export(factorize(x), 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

```
## For demo, a temp. file path is created with the file extension .dta (Stata)
dta_file <- tempfile(fileext = ".dta")
## .csv
csv_file <- tempfile(fileext = ".csv")
## .xlsx
xlsx_file <- tempfile(fileext = ".xlsx")


## Create a Stata data file
export(mtcars, dta_file)

## convert Stata to CSV and open converted file
convert(dta_file, csv_file)
import(csv_file)

## correct an erroneous file format
export(mtcars, xlsx_file, format = "tsv") ## DON'T DO THIS
## import(xlsx_file) ## ERROR
## convert the file by specifying `in_opts`
convert(xlsx_file, xlsx_file, in_opts = list(format = "tsv"))
import(xlsx_file)

## convert from the command line:
## Rscript -e "rio::convert('mtcars.dta', 'mtcars.csv')"
```

---

export *Export*

Description

Write data.frame to a file

Usage

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

Arguments

x	A data frame, matrix or a single-item list of data frame to be written into a file. Exceptions to this rule are that x can be a list of multiple data frames if the output file format is an OpenDocument Spreadsheet (.ods, .fods), 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 base::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 data.table::fwrite()
- Pipe-separated data (.psv), using data.table::fwrite()
- Tab-separated data (.tsv), using data.table::fwrite()
- SAS (.sas7bdat), using haven::write_sas().
- SAS XPORT (.xpt), using haven::write_xpt().
- SPSS (.sav), using haven::write_sav()
- SPSS compressed (.zsav), using haven::write_sav()
- Stata (.dta), using haven::write_dta(). Note that variable/column names containing dots (.) are not allowed and will produce an error.
- Excel (.xlsx), using writexl::write_xlsx(). x can also be a list of data frames; the list entry names are used as sheet names.
- R syntax object (.R), using base::dput() (by default) or base::dump() (if format = 'dump')
- Saved R objects (.RData,.rda), using base::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 `base::saveRDS()`. In this case, `x` can be any serializable R object.
- Serialized R objects (.qs), using `qs::qsave()`, which is significantly faster than .rds. This can be any R object (not just a data frame).
- "XBASE" database files (.dbf), using `foreign::write.dbf()`
- Weka Attribute-Relation File Format (.arff), using `foreign::write.arff()`
- Fixed-width format data (.fwf), using `utils::write.table()` with `row.names = FALSE`, `quote = FALSE`, and `col.names = FALSE`
- CSVY (CSV with a YAML metadata header) using `data.table::fwrite()`.
- Apache Arrow Parquet (.parquet), using `nanoparquet::write_parquet()`
- Feather R/Python interchange format (.feather), using `arrow::write_feather()`
- Fast storage (.fst), using `fst::write.fst()`
- JSON (.json), using `jsonlite::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 `rmatio::write.mat()`
- OpenDocument Spreadsheet (.ods, .fods), using `readODS::write_ods()` or `readODS::write_fods()`.
- HTML (.html), using a custom method based on `xml2::xml_add_child()` to create a simple HTML table and `xml2::write_xml()` to write to disk.
- XML (.xml), using a custom method based on `xml2::xml_add_child()` to create a simple XML tree and `xml2::write_xml()` to write to disk.
- YAML (.yml), using `yaml::write_yaml()`, default to write the content with UTF-8. Might not work on some older systems, e.g. default Windows locale for R <= 4.2.
- Clipboard export (on Windows and Mac OS), using `utils::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

`characterize()`, `import()`, `convert()`, `export_list()`

Examples

```
## For demo, a temp. file path is created with the file extension .csv
csv_file <- tempfile(fileext = ".csv")
## .xlsx
xlsx_file <- tempfile(fileext = ".xlsx")
```

```
## create CSV to import
export(iris, csv_file)

## You can certainly export your data with the file name, which is not a variable:
## import(mtcars, "car_data.csv")

## pass arguments to the underlying function
## data.table::fwrite is the underlying function and `col.names` is an argument
export(iris, csv_file, col.names = FALSE)

## export a list of data frames as worksheets
export(list(a = mtcars, b = iris), xlsx_file)

# NOT RECOMMENDED

## specify `format` to override default format
export(iris, xlsx_file, format = "csv") ## That's confusing
## You can also specify only the format; in the following case
## "mtcars.dta" is written [also confusing]

## export(mtcars, format = "stata")
```

`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, archive = "", ...)
```

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.
`archive`	character. Either empty string (default) to save files in current directory, a path to a (new) directory, or a .zip/.tar file to compress all files into an archive.
`...`	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

```
## For demo, a temp. file path is created with the file extension .xlsx
xlsx_file <- tempfile(fileext = ".xlsx")
export(
    list(
        mtcars1 = mtcars[1:10, ],
        mtcars2 = mtcars[11:20, ],
        mtcars3 = mtcars[21:32, ]
    ),
    xlsx_file
)

# import a single file from multi-object workbook
import(xlsx_file, sheet = "mtcars1")
# import all worksheets, the return value is a list
import_list(xlsx_file)
library('datasets')
export(list(mtcars1 = mtcars[1:10,],
            mtcars2 = mtcars[11:20,],
            mtcars3 = mtcars[21:32,]),
    xlsx_file <- tempfile(fileext = ".xlsx")
)

# import all worksheets
list_of_dfs <- import_list(xlsx_file)

# re-export as separate named files

## export_list(list_of_dfs, file = c("file1.csv", "file2.csv", "file3.csv"))

# re-export as separate files using a name pattern; using the names in the list
## This will be written as "mtcars1.csv", "mtcars2.csv", "mtcars3.csv"

## export_list(list_of_dfs, file = "%s.csv")
```

---

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()`

---

get_info *Get File Info*

Description

A utility function to retrieve the file information of a filename, path, or URL.

Usage

```
get_info(file)

get_ext(file)
```

Arguments

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

Value

For `get_info()`, a list is return with the following slots

- `input` file extension or information used to identify the possible file format
- `format` file format, see `format` argument of `import()`
- `type` "import" (supported by default); "suggest" (supported by suggested packages, see `install_formats()`); "enhance" and "known " are not directly supported; `NA` is unsupported
- `format_name` name of the format
- `import_function` What function is used to import this file
- `export_function` What function is used to export this file
- `file` `file`

For `get_ext()`, just `input` (usually file extension) is returned; retained for backward compatibility.

Examples

```
get_info("starwars.xlsx")
get_info("starwars.ods")
get_info("https://github.com/ropensci/readODS/raw/v2.1/starwars.ods")
get_info("~/duran_duran_rio.mp3")
get_ext("clipboard") ## "clipboard"
get_ext("https://github.com/ropensci/readODS/raw/v2.1/starwars.ods")
```

---

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 = getOption("rio.import.class", "data.frame"),
  which,
  ...
)
```

Arguments

file	A character string naming a file, URL, or single-file (can be Gzip or Bzip2 compressed), .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 tibble), "arrow", "arrow_table" (if using arrow table; the suggested package arrow must be installed) or "data.table" (if using data.table). Other values are ignored, such that a data.frame is returned. The parameter takes precedents over parameters in ... which set a different class.
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 an archive format (zip and tar), 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. But please see the section which below. 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 data.table::fread()
- Pipe-separated data (.psv), using data.table::fread()
- Tab-separated data (.tsv), using data.table::fread()
- SAS (.sas7bdat), using haven::read_sas()
- SAS XPORT (.xpt), using haven::read_xpt()
- SPSS (.sav), using haven::read_sav()
- SPSS compressed (.zsav), using haven::read_sav().
- Stata (.dta), using haven::read_dta()
- SPSS Portable Files (.por), using haven::read_por().
- Excel (.xls and .xlsx), using readxl::read_xlsx() or readxl::read_xls(). Use which to specify a sheet number.

- R syntax object (.R), using `base::dget()`, see `trust` below.
- Saved R objects (.RData,.rda), using `base::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), see `trust` below.
- Serialized R objects (.rds), using `base::readRDS()`. This can be any R object (not just a data frame), see `trust` below.
- Serialized R objects (.qs), using `qs::qread()`, which is significantly faster than .rds. This can be any R object (not just a data frame).
- Epiinfo (.rec), using `foreign::read.epiinfo()`
- Minitab (.mtp), using `foreign::read.mtp()`
- Systat (.syd), using `foreign::read.systat()`
- "XBASE" database files (.dbf), using `foreign::read.dbf()`
- Weka Attribute-Relation File Format (.arff), using `foreign::read.arff()`
- Data Interchange Format (.dif), using `utils::read.DIF()`
- Fortran data (no recognized extension), using `utils::read.fortran()`
- Fixed-width format data (.fwf), using a faster version of `utils::read.fwf()` that requires a `widths` argument and by default in rio has `stringsAsFactors = FALSE`
- CSVY (CSV with a YAML metadata header) using `data.table::fread()`.
- Apache Arrow Parquet (.parquet), using `nanoparquet::read_parquet()`
- Feather R/Python interchange format (.feather), using `arrow::read_feather()`
- Fast storage (.fst), using `fst::read.fst()`
- JSON (.json), using `jsonlite::fromJSON()`
- Matlab (.mat), using `rmatio::read.mat()`
- EViews (.wf1), using `hexView::readEViews()`
- OpenDocument Spreadsheet (.ods, .fods), using `readODS::read_ods()` or `readODS::read_fods()`. Use `which` to specify a sheet number.
- Single-table HTML documents (.html), using `xml2::read_html()`. There is no standard HTML table and we have only tested this with HTML tables exported with this package. HTML tables will only be read correctly if the HTML file can be converted to a list via `xml2::as_list()`. This import feature is not robust, especially for HTML tables in the wild. Please use a proper web scraping framework, e.g. `rvest`.
- Shallow XML documents (.xml), using `xml2::read_xml()`. The data structure will only be read correctly if the XML file can be converted to a list via `xml2::as_list()`.
- YAML (.yml), using `yaml::yaml.load()`
- Clipboard import, using `utils::read.table()` with `row.names = FALSE`
- Google Sheets, as Comma-separated data (.csv)
- GraphPad Prism (.pzfx) using `pzfx::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".

Trust

For serialization formats (.R, .RDS, and .RData), please note that you should only load these files from trusted sources. It is because these formats are not necessarily for storing rectangular data and can also be used to store many things, e.g. code. Importing these files could lead to arbitary code execution. Please read the security principles by the R Project (Plummer, 2024). When importing these files via `rio`, you should affirm that you trust these files, i.e. `trust = TRUE`. See example below. If this affirmation is missing, the current version assumes `trust` to be true for backward compatibility and a deprecation notice will be printed. In the next major release (2.0.0), you must explicitly affirm your trust when importing these files.

Which

For compressed archives (zip and tar, where a compressed file can contain multiple files), it is possible to come to a situation where the parameter `which` is used twice to indicate two different concepts. For example, it is unclear for `.xlsx.zip`whether `which` refers to the selection of an exact file in the archive or the selection of an exact sheet in the decompressed Excel file. In these cases, `rio` assumes that `which` is only used for the selection of file. After the selection of file with `which`, `rio` will return the first item, e.g. the first sheet.

Please note, however, `.gz` and `.bz2` (e.g. `.xlsx.gz`) are compressed, but not archive format. In those cases, `which` is used the same way as the non-compressed format, e.g. selection of sheet for Excel.

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.

References

Plummer, M (2024). Statement on CVE-2024-27322. https://blog.r-project.org/2024/05/10/statement-on-cve-2024-27322/

See Also

`import_list()`, `characterize()`, `gather_attrs()`, `export()`, `convert()`

Examples

```
## For demo, a temp. file path is created with the file extension .csv
csv_file <- tempfile(fileext = ".csv")
## .xlsx
xlsx_file <- tempfile(fileext = ".xlsx")
## create CSV to import
export(iris, csv_file)
## specify `format` to override default format: see export()
export(iris, xlsx_file, format = "csv")

## basic
import(csv_file)

## You can certainly import your data with the file name, which is not a variable:
## import("starwars.csv"); import("mtcars.xlsx")

## Override the default format
## import(xlsx_file) # Error, it is actually not an Excel file
import(xlsx_file, format = "csv")

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

## import CSV as a tibble (or "tbl_df")
import(csv_file, setclass = "tbl_df")

## pass arguments to underlying import function
## data.table::fread is the underlying import function and `nrows` is its argument
import(csv_file, nrows = 20)

## data.table::fread has an argument `data.table` to set the class explicitely to data.table. The
## argument setclass, however, takes precedents over such undocumented features.
class(import(csv_file, setclass = "tibble", data.table = TRUE))

## the default import class can be set with options(rio.import.class = "data.table")
## options(rio.import.class = "tibble"), or options(rio.import.class = "arrow")

## Security
rds_file <- tempfile(fileext = ".rds")
export(iris, rds_file)

## You should only import serialized formats from trusted sources
## In this case, you can trust it because it's generated by you.
import(rds_file, trust = TRUE)
```

`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, compressed directory in a zip file or tar archive, or HTML file)

Usage

```
import_list(
  file,
  setclass = getOption("rio.import.class", "data.frame"),
  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 file, tar archive, 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 tibble), "arrow", "arrow_table" (if using arrow table; the suggested package `arrow` must be installed) or "data.table" (if using data.table). Other values are ignored, such that a data.frame is returned. The parameter takes precedents over parameters in ... which set a different class.
`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 `data.table::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.

Details

When file is a vector of file paths and any files are missing, those files are ignored (with warnings) and this function will not raise any error. For compressed files, the file name must also contain information about the file format of all compressed files, e.g. `files.csv.zip` for this function to work.

Value

If `rbind=FALSE` (the default), a list of a data frames. Otherwise, that list is passed to `data.table::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.

Trust

For serialization formats (.R, .RDS, and .RData), please note that you should only load these files from trusted sources. It is because these formats are not necessarily for storing rectangular data and can also be used to store many things, e.g. code. Importing these files could lead to arbitary code execution. Please read the security principles by the R Project (Plummer, 2024). When importing these files via `rio`, you should affirm that you trust these files, i.e. `trust = TRUE`. See example below. If this affirmation is missing, the current version assumes `trust` to be true for backward compatibility and a deprecation notice will be printed. In the next major release (2.0.0), you must explicitly affirm your trust when importing these files.

Which

For compressed archives (zip and tar, where a compressed file can contain multiple files), it is possible to come to a situation where the parameter `which` is used twice to indicate two different concepts. For example, it is unclear for `.xlsx.zip`whether `which` refers to the selection of an exact file in the archive or the selection of an exact sheet in the decompressed Excel file. In these cases, `rio` assumes that `which` is only used for the selection of file. After the selection of file with `which`, `rio` will return the first item, e.g. the first sheet.

Please note, however, `.gz` and `.bz2` (e.g. `.xlsx.gz`) are compressed, but not archive format. In those cases, `which` is used the same way as the non-compressed format, e.g. selection of sheet for Excel.

References

Plummer, M (2024). Statement on CVE-2024-27322. https://blog.r-project.org/2024/05/10/statement-on-cve-2024-27322/

See Also

`import()`, `export_list()`, `export()`

Examples

```
## For demo, a temp. file path is created with the file extension .xlsx
xlsx_file <- tempfile(fileext = ".xlsx")
export(
    list(
        mtcars1 = mtcars[1:10, ],
        mtcars2 = mtcars[11:20, ],
        mtcars3 = mtcars[21:32, ]
    ),
    xlsx_file
)
```

```
# import a single file from multi-object workbook
import(xlsx_file, sheet = "mtcars1")
# import all worksheets, the return value is a list
import_list(xlsx_file)

# import and rbind all worksheets, the return value is a data frame
import_list(xlsx_file, rbind = TRUE)
```

`install_formats` *Install rio's 'Suggests' Dependencies*

Description

Not all suggested packages are installed by default. These packages are not installed or loaded by default in order to create a slimmer and faster package build, install, and load. Use `show_unsupported_formats()` to check all unsupported formats. `install_formats()` installs all missing 'Suggests' dependencies for rio that expand its support to the full range of support import and export formats.

Usage

```
install_formats(...)

show_unsupported_formats()
```

Arguments

`...`	Additional arguments passed to `utils::install.packages()`.

Value

For `show_unsupported_formats()`, if there is any missing unsupported formats, it return TRUE invisibly; otherwise FALSE. For `install_formats()` it returns TRUE invisibly if the installation is succuessful; otherwise errors.

Examples

```
if (interactive()) {
    install_formats()
}
```

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' dependencies, meaning they are not installed by default. Use `install_formats()` to make sure these packages are available for use.

Author(s)

Maintainer: Chung-hong Chan <chainsawtiney@gmail.com> (ORCID)

Authors:

- Jason Becker <jason@jbecker.co>
- David Schoch <david@schochastics.net> (ORCID)
- Thomas J. Leeper <thosjleeper@gmail.com> (ORCID)

Other contributors:

- Geoffrey CH Chan <gefchchan@gmail.com> [contributor]
- Christopher Gandrud [contributor]
- Andrew MacDonald [contributor]
- Ista Zahn [contributor]
- Stanislaus Stadlmann [contributor]
- Ruaridh Williamson <ruaridh.williamson@gmail.com> [contributor]
- Patrick Kennedy [contributor]
- Ryan Price <ryapric@gmail.com> [contributor]
- Trevor L Davis <trevor.l.davis@gmail.com> [contributor]
- Nathan Day <nathancday@gmail.com> [contributor]
- Bill Denney <wdenney@humanpredictions.com> (ORCID) [contributor]
- Alex Bokov <alex.bokov@gmail.com> (ORCID) [contributor]
- Hugo Gruson (ORCID) [contributor]

References

datamods provides Shiny modules for importing data via `rio`.

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