Package ‘clipr’

October 12, 2022

Type Package
Title Read and Write from the System Clipboard
Version 0.8.0
Description Simple utility functions to read from and write to
the Windows, OS X, and X11 clipboards.
License GPL-3
URL https://github.com/mdlincoln/clipr,
http://matthewlincoln.net/clipr/
BugReports https://github.com/mdlincoln/clipr/issues
Imports utils
Suggests covr, knitr, rmarkdown, rstudioapi (>= 0.5), testthat (>= 2.0.0)
VignetteBuilder knitr
Encoding UTF-8
Language en-US
RoxygenNote 7.1.2
SystemRequirements xclip (https://github.com/astrand/xclip) or xsel
(http://www.vergenet.net/~conrad/software/xsel/) for accessing
the X11 clipboard, or wl-clipboard
(https://github.com/bugaevc/wl-clipboard) for systems using
Wayland.
NeedsCompilation no
Author Matthew Lincoln [aut, cre] (https://orcid.org/0000-0002-4387-3384),
Louis Maddox [ctb],
Steve Simpson [ctb],
Jennifer Bryan [ctb]
Maintainer Matthew Lincoln <matthew.d.lincoln@gmail.com>
Repository CRAN
Date/Publication 2022-02-22 00:58:45 UTC
clear_clip

Description
Clear the system clipboard.

Usage
clear_clip(...)

Arguments
... Pass other options to write_clip().

Note
This is a wrapper function for write_clip(""")

--

clipr

clipr: Read and Write from the System Clipboard

Description
Simple utility functions to read from and write to the Windows, OS X, and X11 clipboards.

Details
The basic functions read_clip() and write_clip() wrap platform-specific functions for writing values from R to the system clipboard. read_clip_tbl() will attempt to process the clipboard content like a table copied from a spreadsheet program. clipr_available() is useful when building packages that depend on clipr functionality.
Description

Checks to see if the system clipboard is write-able/read-able. This may be useful if you are developing a package that relies on clipr and need to ensure that it will skip tests on machines (e.g. CRAN, Travis) where the system clipboard may not be available.

Usage

clipr_available(...)

dr_clipr(...)

Arguments

... Pass other options to write_clip(). Generally only used to pass the argument allow_non_interactive_use = TRUE.

Value

clipr_available returns a boolean value.

Prints an informative message to the console with software and system configuration requirements if clipr is not available (invisibly returns the same string)

Note

This will automatically return FALSE, without even performing the check, if you are running in a non-interactive session. If you must call this non-interactively, be sure to call using clipr_available(allow_non_interactive_use = TRUE), or by setting the environment variable CLIPR_ALLOW=TRUE. Do not attempt to run clipr non-interactively on CRAN; this will result in a failed build!

Examples

## Not run:
# When using testthat:
library(testthat)
skip_if_not(clipr_available())
Description
Read the contents of the system clipboard into a character vector.

Usage
read_clip(allow_non_interactive = Sys.getenv("CLIPR_ALLOW", interactive()))

Arguments
allow_non_interactive
By default, clipr will throw an error if run in a non-interactive session. Set the environment variable CLIPR_ALLOW=TRUE in order to override this behavior.

Value
A character vector with the contents of the clipboard. If the system clipboard is empty, returns NULL

Note
read_clip() will not try to guess at how to parse copied text. If you are copying tabular data, it is suggested that you use read_clip_tbl().

Examples
## Not run:
clip_text <- read_clip()
## End(Not run)

Description
Transforms output of read_clip() into data frame.

Usage
read_clip_tbl(x = read_clip(), ...)

read_clip_tbl
Transforms output of read_clip() into data frame.
write_clip

Arguments

- `x` Defaults to reading from the clipboard, but can be substituted by a character vector already generated by `read_clip()`. The following `read.table()` arguments will be passed by default, but can be overridden by specifying them when calling `read_clip_tbl`:
  - `header` TRUE
  - `sep` "\t"
  - `row.names` 1
  - `stringsAsFactors` FALSE
  - `na.strings` c("NA","")
  - `strip.white` TRUE

- `...` Options to pass to `read.table()`. The following `read.table()` arguments will be passed by default, but can be overridden by specifying them when calling `read_clip_tbl`:

Value

A data frame with the contents of the clipboard. If the system clipboard is empty, returns NULL.

Description

Write a character vector to the system clipboard

Usage

```r
write_clip(
  content,
  object_type = c("auto", "character", "table"),
  breaks = NULL,
  eos = NULL,
  return_new = FALSE,
  allow_non_interactive = Sys.getenv("CLIPR_ALLOW", interactive()),
  ...
)
```

Arguments

- `content` An object to be written to the system clipboard.
- `object_type` `write_clip()` tries to be smart about writing objects in a useful manner. If passed a data.frame or matrix, it will format it using `write.table()` for pasting into an external spreadsheet program. It will otherwise coerce the object to a character vector. `auto` will check the object type, otherwise `table` or `character` can be explicitly specified.
breaks
The separator to be used between each element of the character vector being written. NULL defaults to writing system-specific line breaks between each element of a character vector, or each row of a table.

eos
The terminator to be written after each string, followed by an ASCII nul. Defaults to no terminator character, indicated by NULL.

return_new
If true, returns the rendered string; if false, returns the original object

allow_non_interactive
By default, clipr will throw an error if run in a non-interactive session. Set the environment variable CLIPR_ALLOW=TRUE in order to override this behavior.

... Custom options to be passed to write.table() (if x is a table-like). Defaults to sane line-break and tab standards based on the operating system. By default, this will use col.names = TRUE if the table object has column names, and row.names = TRUE if the object has row names other than c("1", "2", "3"...). Override these defaults by passing arguments here.

Value
Invisibly returns the original object

Note
On X11 systems, write_clip() will cause either xclip (preferred) or xsel to be called. Be aware that, by design, these processes will fork into the background. They will run until the next paste event, when they will then exit silently. (See the man pages for xclip and xsel for more on their behaviors.) However, this means that even if you terminate your R session after running write_clip(), those processes will continue until you access the clipboard via another program. This may be expected behavior for interactive use, but is generally undesirable for non-interactive use. For this reason you must not run write_clip() on CRAN, as the nature of xsel has caused issues in the past.
Call clipr_available() to safely check whether the clipboard is readable and writable.

Examples

```r
## Not run:
text <- "Write to clipboard"
write_clip(text)

multiline <- c("Write", "to", "clipboard")
write_clip(multiline)

# Write
# to
# clipboard

write_clip(multiline, breaks = ",")
# write,to,clipboard

tbl <- data.frame(a=c(1,2,3), b=c(4,5,6))
write_clip(tbl)
```
write_last_clip

## End(Not run)

write_last_clip  

Write contents of the last R expression to the clipboard

### Description
Write contents of the last R expression to the clipboard

### Usage
write_last_clip(...)

### Arguments
...

Pass other options to write_clip().

### Note
This is a wrapper function for write_clip(.Last.value)
Index

clear_clip, 2
clipr, 2
clipr_available, 3
clipr_available(), 2, 6

dr_clipr (clipr_available), 3

read.table(), 4, 5
read_clip, 4
read_clip(), 2, 4, 5
read_clip_tbl, 4
read_clip_tbl(), 2, 4

write.table(), 5, 6
write_clip, 5
write_clip(), 2, 3, 5–7
write_last_clip, 7