Package 'xfun'

June 25, 2019

Type Package

Title Miscellaneous Functions by 'Yihui Xie'

Version 0.8

Description Miscellaneous functions commonly used in other packages maintained by 'Yihui Xie'.

Imports stats, tools

Suggests testit, parallel, rstudioapi, tinytex, mime, markdown, knitr, htmltools, base64enc, remotes, rmarkdown

License MIT + file LICENSE

URL https://github.com/yihui/xfun

BugReports https://github.com/yihui/xfun/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Yihui Xie [aut, cre, cph] (<https://orcid.org/0000-0003-0645-5666>), Daijiang Li [ctb], Xianying Tan [ctb]

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2019-06-25 05:50:03 UTC

R topics documented:

`attr` *Obtain an attribute of an object without partial matching*

Description

An abbreviation of `base::attr(exact = TRUE)`.

Usage

```
attr(...)
```

Arguments

`...`	Passed to `base::attr()` (without the `exact` argument).

Examples

```
z = structure(list(a = 1), foo = 2)
base::attr(z, "f")  # 2
xfun::attr(z, "f")  # NULL
xfun::attr(z, "foo")  # 2
```

---

download_file *Try various methods to download a file*

Description

Try all possible methods in `download.file()` (e.g., `libcurl`, `curl`, `wget`, and `wininet`) and see if any method can succeed. The reason to enumerate all methods is that sometimes the default method does not work, e.g., https://stat.ethz.ch/pipermail/r-devel/2016-June/072852.html.

Usage

```
download_file(url, output = basename(url), ...)
```

Arguments

`url`	The URL of the file.
`output`	Path to the output file. If not provided, the base name of the URL will be used (query parameters and hash in the URL will be removed).
`...`	Other arguments to be passed to `download.file()` (except `method`).

Value

The integer code `0` for success, or an error if none of the methods work.

---

embed_file *Embed a file, multiple files, or directory on an HTML page*

Description

For a file, first encode it into base64 data (a character string). Then generate a hyperlink of the form '`<a href="base64 data" download="filename">Download filename</a>`'. The file can be downloaded when the link is clicked in modern web browsers. For a directory, it will be compressed as a zip archive first, and the zip file is passed to `embed_file()`. For multiple files, they are also compressed to a zip file first.

Usage

```
embed_file(path, name = basename(path), text = paste("Download", name), ...)

embed_dir(path, name = paste0(normalize_path(path), ".zip"), ...)

embed_files(path, name = with_ext(basename(path[1]), ".zip"), ...)
```

Arguments

`path`	Path to the file(s) or directory.
`name`	The default filename to use when downloading the file. Note that for `embed_dir()`, only the base name (of the zip filename) will be used.
`text`	The text for the hyperlink.
`...`	For `embed_file()`, additional arguments to be passed to `htmltools::a()` (e.g., `class = 'foo'`). For `embed_dir()` and `embed_files()`, arguments passed to `embed_file()`.

Details

These functions can be called in R code chunks in R Markdown documents with HTML output formats. You may embed an arbitrary file or directory in the HTML output file, so that readers of the HTML page can download it from the browser. A common use case is to embed data files for readers to download.

Value

An HTML tag '<a>' with the appropriate attributes.

Note

Windows users may need to install Rtools to obtain the `zip` command to use `embed_dir()` and `embed_files()`.

These functions require R packages **mime**, **base64enc**, and **htmltools**. If you have installed the **rmarkdown** package, these packages should be available, otherwise you need to install them separately.

Currently Internet Explorer does not support downloading embedded files (https://caniuse.com/#feat=download).

Examples

```
logo = file.path(R.home("doc"), "html", "logo.jpg")
link = xfun::embed_file(logo, "R-logo.jpg", "Download R logo")
link
htmltools::browsable(link)
```

`file_ext` *Manipulate filename extensions*

Description

Functions to obtain (`file_ext()`), remove (`sans_ext()`), and change (`with_ext()`) extensions in filenames.

Usage

```
file_ext(x)

sans_ext(x)

with_ext(x, ext)
```

Arguments

`x`	A character of file paths.
`ext`	A vector of new extensions.

Details

`file_ext()` is a wrapper of `tools::file_ext()`. `sans_ext()` is a wrapper of `tools::file_path_sans_ext()`.

Value

A character vector of the same length as `x`.

Examples

```
library(xfun)
p = c("abc.doc", "def123.tex", "path/to/foo.Rmd")
file_ext(p)
sans_ext(p)
with_ext(p, ".txt")
with_ext(p, c(".ppt", ".sty", ".Rnw"))
with_ext(p, "html")
```

file_string *Read a text file and concatenate the lines by '\n'*

Description

The source code of this function should be self-explanatory.

Usage

```
file_string(file)
```

Arguments

file	Path to a text file (should be encoded in UTF-8).

Value

A character string of text lines concatenated by '\n'.

Examples

```
xfun::file_string(system.file("DESCRIPTION", package = "xfun"))
```

gsub_file *Search and replace strings in files*

Description

These functions provide the "file" version of gsub(), i.e., they perform searching and replacement in files via gsub().

Usage

```
gsub_file(file, ..., rw_error = TRUE)

gsub_files(files, ...)

gsub_dir(..., dir = ".", recursive = TRUE, ext = NULL, mimetype = ".*")

gsub_ext(ext, ..., dir = ".", recursive = TRUE)
```

Arguments

file	Path of a single file.
...	For gsub_file(), arguments passed to gsub(). For other functions, arguments passed to gsub_file(). Note that the argument x of gsub() is the content of the file.
rw_error	Whether to signal an error if the file cannot be read or written. If FALSE, the file will be ignored (with a warning).
files	A vector of file paths.
dir	Path to a directory (all files under this directory will be replaced).
recursive	Whether to find files recursively under a directory.
ext	A vector of filename extensions (without the leading periods).
mimetype	A regular expression to filter files based on their MIME types, e.g., '^text/' for plain text files. This requires the **mime** package.

Note

These functions perform in-place replacement, i.e., the files will be overwritten. Make sure you backup your files in advance, or use version control!

Examples

```
library(xfun)
f = tempfile()
writeLines(c("hello", "world"), f)
gsub_file(f, "world", "woRld", fixed = TRUE)
readLines(f)
```

---

install_dir *Install a source package from a directory*

Description

Run R CMD build to build a tarball from a source directory, and run R CMD INSTALL to install it.

Usage

```
install_dir(src, build = TRUE, build_opts = NULL, install_opts = NULL)
```

Arguments

src	The package source directory.
build	Whether to build a tarball from the source directory. If FALSE, run R CMD INSTALL on the directory directly (note that vignettes will not be automatically built).
build_opts	The options for R CMD build.
install_opts	The options for R CMD INSTALL.

Value

Invisible status from `R CMD INSTALL`.

---

install_github *An alias of* `remotes::install_github()`

Description

This alias is to make autocomplete faster via `xfun::install_github`, because most `remotes::install_*` functions are never what I want. I only use `install_github` and it is inconvenient to autocomplete it, e.g. `install_git` always comes before `install_github`, but I never use it. In RStudio, I only need to type `xfun::ig` to get `xfun::install_github`.

Usage

```
install_github(...)
```

Arguments

`...`	Arguments to be passed to `remotes::install_github()`.

---

in_dir *Evaluate an expression under a specified working directory*

Description

Change the working directory, evaluate the expression, and restore the working directory.

Usage

```
in_dir(dir, expr)
```

Arguments

`dir`	Path to a directory.
`expr`	An R expression.

Examples

```
library(xfun)
in_dir(tempdir(), {
    print(getwd())
    list.files()
})
```

isFALSE *Test if an object is identical to* FALSE

Description

A simple abbreviation of `identical(x, FALSE)`.

Usage

```
isFALSE(x)
```

Arguments

x	An R object.

Examples

```
library(xfun)
isFALSE(TRUE)  # false
isFALSE(FALSE)  # true
isFALSE(c(FALSE, FALSE))  # false
```

is_ascii *Check if a character vector consists of entirely ASCII characters*

Description

Converts the encoding of a character vector to `'ascii'`, and check if the result is `NA`.

Usage

```
is_ascii(x)
```

Arguments

x	A character vector.

Value

A logical vector indicating whether each element of the character vector is ASCII.

Examples

```
library(xfun)
is_ascii(letters)  # yes
is_ascii(intToUtf8(8212))  # no
```

is_windows *Test for types of operating systems*

Description

Functions based on `.Platform$OS.type` and `Sys.info()` to test if the current operating system is Windows, macOS, Unix, or Linux.

Usage

```
is_windows()

is_unix()

is_macos()

is_linux()
```

Examples

```
library(xfun)
# only one of the following statements should be true
is_windows()
is_unix() && is_macos()
is_linux()
```

native_encode *Try to use the system native encoding to represent a character vector*

Description

Apply `enc2native()` to the character vector, and check if `enc2utf8()` can convert it back without a loss. If it does, return `enc2native(x)`, otherwise return the original vector with a warning.

Usage

```
native_encode(x, windows_only = is_windows())
```

Arguments

`x`	A character vector.
`windows_only`	Whether to make the attempt on Windows only. On Unix, characters are typically encoded in the native encoding (UTF-8), so there is no need to do the conversion.

Examples

```
library(xfun)
s = intToUtf8(c(20320, 22909))
Encoding(s)

s2 = native_encode(s)
Encoding(s2)
```

---

normalize_path *Normalize paths*

Description

A wrapper function of `normalizePath()` with different defaults.

Usage

```
normalize_path(path, winslash = "/", must_work = FALSE)
```

Arguments

`path, winslash, must_work`
: Arguments passed to `normalizePath()`.

Examples

```
library(xfun)
normalize_path("~")
```

---

numbers_to_words *Convert numbers to English words*

Description

This can be helpful when writing reports with **knitr/rmarkdown** if we want to print numbers as English words in the output. The function `n2w()` is an alias of `numbers_to_words()`.

Usage

```
numbers_to_words(x, cap = FALSE, hyphen = TRUE, and = FALSE)

n2w(x, cap = FALSE, hyphen = TRUE, and = FALSE)
```

Arguments

x	A numeric vector. Values should be integers. The absolute values should be less than 1e15.
cap	Whether to capitalize the first letter of the word. This can be useful when the word is at the beginning of a sentence. Default is FALSE.
hyphen	Whether to insert hyphen (-) when the number is between 21 and 99 (except 30, 40, etc.).
and	Whether to insert and between hundreds and tens, e.g., write 110 as "one hundred and ten" if TRUE instead of "one hundred ten".

Value

A character vector.

Author(s)

Daijiang Li

Examples

```
library(xfun)
n2w(0, cap = TRUE)
n2w(0:121, and = TRUE)
n2w(1e+06)
n2w(1e+11 + 12345678)
n2w(-987654321)
n2w(1e+15 - 1)
```

---

optipng *Run OptiPNG on all PNG files under a directory*

Description

Calls the command optipng to optimize all PNG files under a directory.

Usage

```
optipng(dir = ".")
```

Arguments

dir	Path to a directory.

References

OptiPNG: http://optipng.sourceforge.net.

parse_only *Parse R code and do not keep the source*

Description

An abbreviation of `parse(keep.source = FALSE)`.

Usage

```
parse_only(code)
```

Arguments

code	A character vector of the R source code.

Value

R expressions.

Examples

```
library(xfun)
parse_only("1+1")
parse_only(c("y~x", "1:5 # a comment"))
parse_only(character(0))
```

pkg_attach *Attach or load packages, and automatically install missing packages if requested*

Description

`pkg_attach()` is a vectorized version of `library()` over the package argument to attach multiple packages in a single function call. `pkg_load()` is a vectorized version of `requireNamespace()` to load packages (without attaching them). The functions `pkg_attach2()` and `pkg_load2()` are wrappers of `pkg_attach(install = TRUE)` and `pkg_load(install = TRUE)`, respectively. `loadable()` is an abbreviation of `requireNamespace(quietly = TRUE)`.

Usage

```
pkg_attach(..., install = FALSE, message = getOption("xfun.pkg_attach.message", TRUE))

pkg_load(..., error = TRUE, install = FALSE)

loadable(pkg, strict = TRUE, new_session = FALSE)
```

```
pkg_attach2(...)

pkg_load2(...)
```

Arguments

`...`	Package names (character vectors, and must always be quoted).
`install`	Whether to automatically install packages that are not available using `install.packages()`. You are recommended to set a CRAN mirror in the global option `repos` via `options()` if you want to automatically install packages.
`message`	Whether to show the package startup messages (if any startup messages are provided in a package).
`error`	Whether to signal an error when certain packages cannot be loaded.
`pkg`	A single package name.
`strict`	If `TRUE`, use `requireNamespace()` to test if a package is loadable; otherwise only check if the package is in `.packages(TRUE)` (this does not really load the package, so it is less rigorous but on the other hand, it can keep the current R session clean).
`new_session`	Whether to test if a package is loadable in a new R session. Note that `new_session = TRUE` implies `strict = TRUE`.

Details

These are convenience functions that aim to solve these common problems: (1) We often need to attach or load multiple packages, and it is tedious to type several `library()` calls; (2) We are likely to want to install the packages when attaching/loading them but they have not been installed.

Value

`pkg_attach()` returns `NULL` invisibly. `pkg_load()` returns a logical vector, indicating whether the packages can be loaded.

Examples

```
library(xfun)
pkg_attach("stats", "graphics")
# pkg_attach2('servr') # automatically install servr if it is not installed

(pkg_load("stats", "graphics"))
```

prose_index *Find the indices of lines in Markdown that are prose (not code blocks)*

Description

Filter out the indices of lines between code block fences such as ``` (could be three or four or more backticks).

Usage

```
prose_index(x, warn = TRUE)
```

Arguments

x	A character vector of text in Markdown.
warn	Whether to emit a warning when code fences are not balanced.

Value

An integer vector of indices of lines that are prose in Markdown.

Note

If the code fences are not balanced (e.g., a starting fence without an ending fence), this function will treat all lines as prose.

Examples

```
library(xfun)
prose_index(c("a", "````", "b", "````", "c"))
prose_index(c("a", "`````", "```r", "1+1", "```", "`````", "c"))
```

protect_math *Protect math expressions in pairs of backticks in Markdown*

Description

For Markdown renderers that do not support LaTeX math, we need to protect math expressions as verbatim code (in a pair of backticks), because some characters in the math expressions may be interpreted as Markdown syntax (e.g., a pair of underscores may make text italic). This function detects math expressions in Markdown (by heuristics), and wrap them in backticks.

Usage

```
protect_math(x)
```

Arguments

x	A character vector of text in Markdown.

Details

Expressions in pairs of dollar signs or double dollar signs are treated as math, if there are no spaces after the starting dollar sign, or before the ending dollar sign. There should be spaces before the starting dollar sign, unless the math expression starts from the very beginning of a line. For a pair of single dollar signs, the ending dollar sign should not be followed by a number. With these assumptions, there should not be too many false positives when detecing math expressions.

Besides, LaTeX environments (\begin{*} and \end{*}) are also protected in backticks.

Value

A character vector with math expressions in backticks.

Note

If you are using Pandoc or the **rmarkdown** package, there is no need to use this function, because Pandoc's Markdown can recognize math expressions.

Examples

```
library(xfun)
protect_math(c("hi $a+b$", "hello $$\\alpha$$", "no math here: $x is $10 dollars"))
protect_math(c("hi $$", "\\begin{equation}", "x + y = z", "\\end{equation}"))
```

---

raw_string *Print a character vector in its raw form*

Description

The function `raw_string()` assigns the class `xfun_raw_string` to the character vector, and the corresponding printing function `print.xfun_raw_string()` uses `cat(x, sep = '\n')` to write the character vector to the console, which will suppress the leading indices (such as `[1]`) and double quotes, and it may be easier to read the characters in the raw form (especially when there are escape sequences).

Usage

```
raw_string(x)

## S3 method for class 'xfun_raw_string'
print(x, ...)
```

Arguments

x	For raw_string(), a character vector. For the print method, the raw_string() object.
...	Other arguments (currently ignored).

Examples

```
library(xfun)
raw_string(head(LETTERS))
raw_string(c("a \"b\"", "hello\tworld!"))
```

---

read_utf8 *Read / write files encoded in UTF-8*

Description

Read or write files, assuming they are encoded in UTF-8. read_utf8() is roughly readLines(encoding = 'UTF-8') (a warning will be issued if non-UTF8 lines are found), and write_utf8() calls writeLines(enc2utf8(text), useBytes =

Usage

```
read_utf8(con, error = FALSE)

write_utf8(text, con, ...)
```

Arguments

con	A connection or a file path.
error	Whether to signal an error when non-UTF8 characters are detected (if FALSE, only a warning message is issued).
text	A character vector (will be converted to UTF-8 via enc2utf8()).
...	Other arguments passed to writeLines() (except useBytes, which is TRUE in write_utf8()).

`rev_check` *Run* `R CMD check` *on the reverse dependencies of a package*

Description

Install the source package, figure out the reverse dependencies on CRAN, download all of their source packages, and run `R CMD check` on them in parallel.

Usage

```
rev_check(pkg, which = "all", recheck = FALSE, ignore = NULL, update = TRUE,
   src = file.path(src_dir, pkg), src_dir = getOption("xfun.rev_check.src_dir"))

compare_Rcheck(status_only = FALSE, output = "00check_diffs.md")
```

Arguments

`pkg`	The package name.
`which`	Which types of reverse dependencies to check. See `tools::package_dependencies()` for possible values. The special value `'hard'` means the hard dependencies, i.e., `c('Depends', 'Imports', 'LinkingTo')`.
`recheck`	Whether to only check the failed packages from last time. By default, if there are any '`*.Rcheck`' directories, `recheck` will be automatically set to `TRUE` if missing.
`ignore`	A vector of package names to be ignored in `R CMD check`. If this argument is missing and a file '`00ignore`' exists, the file will be read as a character vector and passed to this argument.
`update`	Whether to update all packages before the check.
`src`	The path of the source package directory.
`src_dir`	The parent directory of the source package directory. This can be set in a global option if all your source packages are under a common parent directory.
`status_only`	If `TRUE`, only compare the final statuses of the checks (the last line of '`00check.log`'), and delete '`*.Rcheck`' and '`*.Rcheck2`' if the statuses are identical, otherwise write out the full diffs of the logs. If `FALSE`, compare the full logs under '`*.Rcheck`' and '`*.Rcheck2`'.
`output`	The output Markdown file to which the diffs in check logs will be written. If the **markdown** package is available, the Markdown file will be converted to HTML, so you can see the diffs more clearly.

Details

Everything occurs under the current working directory, and you are recommended to call this function under a designated directory, especially when the number of reverse dependencies is large, because all source packages will be downloaded to this directory, and all '`*.Rcheck`' directories will be generated under this directory, too.

If a source tarball of the expected version has been downloaded before (under the 'tarball' directory), it will not be downloaded again (to save time and bandwidth).

After a package has been checked, the associated '*.Rcheck' directory will be deleted if the check was successful (no warnings or errors or notes), which means if you see a '*.Rcheck' directory, it means the check failed, and you need to take a look at the log files under that directory.

The time to finish the check is recorded for each package. As the check goes on, the total remaing time will be roughly estimated via `n * mean(times)`, where `n` is the number of packages remaining to be checked, and `times` is a vector of elapsed time of packages that have been checked.

If a check on a reverse dependency failed, its '*.Rcheck' directory will be renamed to '*.Rcheck2', and another check will be run against the CRAN version of the package. If the logs of the two checks are the same, it means no new problems were introduced in the package, and you can probably ignore this particular reverse dependency. The function `compare_Rcheck()` can be used to create a summary of all the differences in the check logs under '*.Rcheck' and '*.Rcheck2'. This will be done automatically if `options(xfun.rev_check.summary = TRUE)` has been set.

A recommended workflow is to use a special directory to run `rev_check()`, set the global `options` `xfun.rev_check.src_dir` and `repos` in the R startup (see `?Startup`) profile file `.Rprofile` under this directory, and (optionally) set `R_LIBS_USER` in '`.Renviron`' to use a special library path (so that your usual library will not be cluttered). Then run `xfun::rev_check(pkg)` once, investigate and fix the problems or (if you believe it was not your fault) ignore broken packages in the file '`00ignore`', and run `xfun::rev_check(pkg)` again to recheck the failed packages. Repeat this process until all '*.Rcheck' directories are gone.

As an example, I set `options(repos = c(CRAN = 'https://cran.rstudio.com'), xfun.rev_check.src_dir = '~/Dr` in '`.Rprofile`', and `R_LIBS_USER=~/R-tmp` in '`.Renviron`'. Then I can run, for example, `xfun::rev_check('knitr')` repeatedly under a special directory '`~/Downloads/revcheck`'. Reverse dependencies and their dependencies will be installed to '`~/R-tmp`', and **knitr** will be installed from '`~/Dropbox/repo/kintr`'.

See Also

`devtools::revdep_check()` is more sophisticated, but currently has a few major issues that affect me: (1) It always deletes the '*.Rcheck' directories (https://github.com/hadley/devtools/issues/1395), which makes it difficult to know more information about the failures; (2) It does not fully install the source package before checking its reverse dependencies (https://github.com/hadley/devtools/pull/1397); (3) I feel it is fairly difficult to iterate the check (ignore the successful packages and only check the failed packages); by comparison, `xfun::rev_check()` only requires you to run a short command repeatedly (failed packages are indicated by the existing '*.Rcheck' directories, and automatically checked again the next time).

`xfun::rev_check()` borrowed a very nice feature from `devtools::revdep_check()`: estimating and displaying the remaining time. This is particularly useful for packages with huge numbers of reverse dependencies.

`Rscript` *Run the commands* `Rscript` *and* `R CMD`

Description

Wrapper functions to run the commands `Rscript` and `R CMD`.

Usage

```
Rscript(args, ...)

Rcmd(args, ...)
```

Arguments

args	A character vector of command-line arguments.
...	Other arguments to be passed to system2().

Value

A value returned by system2().

Examples

```
library(xfun)
Rscript(c("-e", "1+1"))
Rcmd(c("build", "--help"))
```

---

rstudio_type *Type a character vector into the RStudio source editor*

Description

Use the **rstudioapi** package to insert characters one by one into the RStudio source editor, as if they were typed by a human.

Usage

```
rstudio_type(x, pause = function() 0.1, mistake = 0, save = 0)
```

Arguments

x	A character vector.
pause	A function to return a number in seconds to pause after typing each character.
mistake	The probability of making random mistakes when typing the next character. A random mistake is a random string typed into the editor and deleted immediately.
save	The probability of saving the document after typing each character. Note that If a document is not opened from a file, it will never be saved.

Examples

```
library(xfun)
if (loadable("rstudioapi") && rstudioapi::isAvailable()) {
    rstudio_type("Hello, RStudio! xfun::rstudio_type() looks pretty cool!",
        pause = function() runif(1, 0, 0.5), mistake = 0.1)
}
```

same_path *Test if two paths are the same after they are normalized*

Description

Compare two paths after normalizing them with the same separator (/).

Usage

```
same_path(p1, p2, ...)
```

Arguments

p1, p2	Two vectors of paths.
...	Arguments to be passed to normalize_path().

Examples

```
library(xfun)
same_path("~/foo", file.path(Sys.getenv("HOME"), "foo"))
```

session_info *An alternative to sessionInfo() to print session information*

Description

This function tweaks the output of sessionInfo(): (1) It adds the RStudio version information if running in the RStudio IDE; (2) It removes the information about matrix products, BLAS, and LAPACK; (3) It removes the names of base R packages; (4) It prints out package versions in a single group, and does not differentiate between loaded and attached packages.

Usage

```
session_info(packages = NULL, dependencies = TRUE)
```

Arguments

packages	A character vector of package names, of which the versions will be printed. If not specified, it means all loaded and attached packages in the current R session.
dependencies	Whether to print out the versions of the recursive dependencies of packages.

Details

It also allows you to only print out the versions of specified packages (via the `packages` argument) and optionally their recursive dependencies. For these specified packages (if provided), if a function `xfun_session_info()` exists in a package, it will be called and expected to return a character vector to be appended to the output of `session_info()`. This provides a mechanism for other packages to inject more information into the `session_info` output. For example, **rmarkdown** (>= 1.20.2) has a function `xfun_session_info()` that returns the version of Pandoc, which can be very useful information for diagnostics.

Value

A character vector of the session information marked as `raw_string()`.

Examples

```
xfun::session_info()
if (loadable("MASS")) xfun::session_info("MASS")
```

`strict_list` *Strict lists*

Description

A strict list is essentially a normal `list()` but it does not allow partial matching with `$`.

Usage

```
strict_list(...)

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

## S3 method for class 'xfun_strict_list'
print(x, ...)
```

Arguments

`...`	Objects (list elements), possibly named. Ignored in the `print()` method.
`x`	A strict list.
`name`	The name (a character string) of the list element.

Details

To me, partial matching is often more annoying and surprising than convenient. It can lead to bugs that are very hard to discover, and I have been bitten for many times. When I write `x$name`, I always mean precisely `name`. You should use a modern code editor to autocomplate the `name` if it is too long to type, instead of using a partial name.

Value

`strict_list()` returns a list with the class `xfun_strict_list`.

Examples

```
library(xfun)
(z = strict_list(aaa = "I am aaa", b = 1:5))
z$a  # NULL!
z$aaa  # I am aaa
z$b
z$c = "create a new element"

z2 = unclass(z)  # a normal list
z2$a  # partial matching
```

---

stringsAsStrings *Set the global option* `options(stringsAsFactors = FALSE)` *inside a parent function and restore the option after the parent function exits*

Description

This is a shorthand of `opts = options(stringsAsFactors = FALSE); on.exit(options(opts), add = TRUE)`; `strings_please()` is an alias of `stringsAsStrings()`.

Usage

```
stringsAsStrings()

strings_please()
```

Examples

```
f = function() {
    xfun::strings_please()
    data.frame(x = letters[1:4], y = factor(letters[1:4]))
}
str(f())  # the first column should be character
```

`tojson` *A simple JSON serializer*

Description

A JSON serializer that only works on a limited types of R data (`NULL`, lists, logical scalars, character/numeric vectors). A character string of the class `JS_EVAL` is treated as raw JavaScript, so will not be quoted. The function `json_vector()` converts an atomic R vector to JSON.

Usage

```
tojson(x)

json_vector(x, to_array = FALSE, quote = TRUE)
```

Arguments

`x`	An R object.
`to_array`	Whether to convert a vector to a JSON array (use `[]`).
`quote`	Whether to double quote the elements.

Value

A character string.

See Also

The **jsonlite** package provides a full JSON serializer.

Examples

```
library(xfun)
tojson(NULL)
tojson(1:10)
tojson(TRUE)
tojson(FALSE)
cat(tojson(list(a = 1, b = list(c = 1:3, d = "abc"))))
cat(tojson(list(c("a", "b"), 1:5, TRUE)))

# the class JS_EVAL is originally from htmlwidgets::JS()
JS = function(x) structure(x, class = "JS_EVAL")
cat(tojson(list(a = 1:5, b = JS("function() {return true;}"))))
```

try_silent *Try to evaluate an expression silently*

Description

An abbreviation of `try(silent = TRUE)`.

Usage

```
try_silent(expr)
```

Arguments

`expr`	An R expression.

Examples

```
library(xfun)
z = try_silent(stop("Wrong!"))
inherits(z, "try-error")
```

upload_ftp *Upload to an FTP server via* `curl`

Description

Run the command `curl -T file server` to upload a file to an FTP server. These functions require the system package (*not the R package*) `curl` to be installed (which should be available on macOS by default). The function `upload_win_builder()` uses `upload_ftp()` to upload packages to the win-builder server.

Usage

```
upload_ftp(file, server, dir = "")

upload_win_builder(file, version = c("R-devel", "R-release", "R-oldrelease"),
    server = "ftp://win-builder.r-project.org/")
```

Arguments

`file`	Path to a local file.
`server`	The address of the FTP server.
`dir`	The remote directory to which the file should be uploaded.
`version`	The R version(s) on win-builder.

Details

These functions were written mainly to save package developers the trouble of going to the win-builder web page and uploading packages there manually. You may also consider using `devtools::check_win_*`, which currently only allows you to upload a package to one folder on win-builder each time, and `xfun::upload_win_builder()` uploads to all three folders, which is more likely to be what you need.

Value

Status code returned from `system2`.

Index