Package: withr (via r-universe)

June 17, 2024

Title Run Code 'With' Temporarily Modified Global State

Version 3.0.0.9000

Description A set of functions to run code 'with' safely and temporarily modified global state. Many of these functions were originally a part of the 'devtools' package, this provides a simple package with limited dependencies to provide access to these functions.

License MIT + file LICENSE

URL https://withr.r-lib.org, https://github.com/r-lib/withr#readme

BugReports https://github.com/r-lib/withr/issues

Depends R (>= 3.5.0)

Imports graphics, grDevices,

Suggests callr, covr, DBI, knitr, lattice, methods, rlang, rmarkdown (>= 2.12), RSQLite, testthat (>= 3.0.0)

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.0.9000

Collate 'aaa.R' 'collate.R' 'connection.R' 'db.R' 'defer-exit.R' 'standalone-defer.R' 'defer.R' 'wrap.R' 'local_.R' 'with_.R' 'devices.R' 'dir.R' 'env.R' 'file.R' 'language.R' 'libpaths.R' 'locale.R' 'makevars.R' 'namespace.R' 'options.R' 'par.R' 'path.R' 'rng.R' 'seed.R' 'sink.R' 'tempfile.R' 'timezone.R' 'torture.R' 'utils.R' 'with.R'

Config/testthat/edition 3

Config/Needs/website tidyverse/tidytemplate

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

RemoteUrl https://github.com/r-lib/withr

RemoteRef HEAD

RemoteSha 334415234c4b049d125c1a0b47a20d682daef9ad

Contents

defer 2
devices 4
withr 10
with_collate 12
with_connection 13
with_db_connection 14
with_dir 15
with_envvar 16
with_file 17
with_gctorture2 18
with_language 19
with_libpaths 19
with_locale 20
with_makevars 22
with_options 23
with_package 24
with_par 26
with_path 28
with_rng_version 29
with_seed 30
with_sink 31
with_tempfile 32
with_temp_libpaths 34
with_timezone 35

Index **37**

`defer` *Defer Evaluation of an Expression*

Description

Similar to `on.exit()`, but allows one to attach an expression to be evaluated when exiting any frame currently on the stack. This provides a nice mechanism for scoping side effects for the duration of a function's execution.

Usage

```
defer(expr, envir = parent.frame(), priority = c("first", "last"))

defer_parent(expr, priority = c("first", "last"))

deferred_run(envir = parent.frame())

deferred_clear(envir = parent.frame())
```

Arguments

`expr`	`[expression]` An expression to be evaluated.
`envir`	`[environment]` Attach exit handlers to this environment. Typically, this should be either the current environment or a parent frame (accessed through `parent.frame()`).
`priority`	`[character(1)]` Specify whether this handler should be executed `"first"` or `"last"`, relative to any other registered handlers on this environment.

Details

`defer()` works by attaching handlers to the requested environment (as an attribute called `"handlers"`), and registering an exit handler that executes the registered handler when the function associated with the requested environment finishes execution.

Deferred events can be set on the global environment, primarily to facilitate the interactive development of code that is intended to be executed inside a function or test. A message alerts the user to the fact that an explicit `deferred_run()` is the only way to trigger these deferred events. Use `deferred_clear()` to clear them without evaluation. The global environment scenario is the main motivation for these functions.

Running handlers within `source()`

withr handlers run within `source()` are run when `source()` exits rather than line by line.

This is only the case when the script is sourced in `globalenv()`. For a local environment, the caller needs to set `options(withr.hook_source = TRUE)`. This is to avoid paying the penalty of detecting `source()` in the normal usage of `defer()`.

Examples

```
# define a 'local' function that creates a file, and
# removes it when the parent function has finished executing
local_file <- function(path) {
  file.create(path)
  defer_parent(unlink(path))
}

# create tempfile path
path <- tempfile()

# use 'local_file' in a function
local({
  local_file(path)
  stopifnot(file.exists(path))
})

# file is deleted as we leave 'local' local
stopifnot(!file.exists(path))
```

```
# investigate how 'defer' modifies the
# executing function's environment
local({
  local_file(path)
  print(attributes(environment()))
})

# Note that examples lack function scoping so deferred calls are
# generally executed immediately
defer(print("one"))
defer(print("two"))
```

devices *Graphics devices*

Description

Temporarily use a graphics device.

Usage

```
with_bmp(new, code, ...)

local_bmp(new = list(), ..., .local_envir = parent.frame())

with_cairo_pdf(new, code, ...)

local_cairo_pdf(new = list(), ..., .local_envir = parent.frame())

with_cairo_ps(new, code, ...)

local_cairo_ps(new = list(), ..., .local_envir = parent.frame())

with_pdf(
  new,
  code,
  width,
  height,
  onefile,
  family,
  title,
  fonts,
  version,
  paper,
  encoding,
  bg,
  fg,
  pointsize,
  pagecentre,
  colormodel,
  useDingbats,
  useKerning,
  fillOddEven,
  compress
)

local_pdf(
  new = list(),
  width,
  height,
  onefile,
  family,
  title,
  fonts,
  version,
  paper,
  encoding,
  bg,
  fg,
  pointsize,
  pagecentre,
  colormodel,
  useDingbats,
  useKerning,
  fillOddEven,
  compress,
  .local_envir = parent.frame()
)

with_postscript(
  new,
  code,
  onefile,
  family,
  title,
  fonts,
  encoding,
  bg,
  fg,
  width,
  height,
  horizontal,
  pointsize,
  paper,
  pagecentre,
  print.it,
  command,
  colormodel,
  useKerning,
  fillOddEven
)

local_postscript(
  new = list(),
  onefile,
  family,
  title,
  fonts,
  encoding,
  bg,
  fg,
  width,
  height,
  horizontal,
  pointsize,
  paper,
  pagecentre,
  print.it,
  command,
  colormodel,
  useKerning,
  fillOddEven,
  .local_envir = parent.frame()
)

with_svg(
  new,
  code,
  width = 7,
  height = 7,
  pointsize = 12,
  onefile = FALSE,
  family = "sans",
  bg = "white",
  antialias = c("default", "none", "gray", "subpixel"),
  ...
)

local_svg(
  new = list(),
  width = 7,
  height = 7,
  pointsize = 12,
  onefile = FALSE,
  family = "sans",
  bg = "white",
  antialias = c("default", "none", "gray", "subpixel"),
  ...,
  .local_envir = parent.frame()
)

with_tiff(new, code, ...)

local_tiff(new = list(), ..., .local_envir = parent.frame())

with_xfig(
  new,
  code,
  onefile = FALSE,
  encoding = "none",
  paper = "default",
  horizontal = TRUE,
  width = 0,
  height = 0,
  family = "Helvetica",
  pointsize = 12,
  bg = "transparent",
  fg = "black",
  pagecentre = TRUE,
  defaultfont = FALSE,
  textspecial = FALSE
)

local_xfig(
  new = list(),
  onefile = FALSE,
  encoding = "none",
  paper = "default",
  horizontal = TRUE,
  width = 0,
  height = 0,
  family = "Helvetica",
  pointsize = 12,
  bg = "transparent",
  fg = "black",
  pagecentre = TRUE,
  defaultfont = FALSE,
  textspecial = FALSE,
  .local_envir = parent.frame()
)

with_png(new, code, ...)
```

```
local_png(new = list(), ..., .local_envir = parent.frame())

with_jpeg(new, code, ...)

local_jpeg(new = list(), ..., .local_envir = parent.frame())
```

Arguments

`new`	`[named character]` New graphics device
`code`	`[any]` Code to execute in the temporary environment
`...`	Additional arguments passed to the graphics device.
`.local_envir`	`[environment]` The environment to use for scoping.
`width`	the width of the device in inches.
`height`	the height of the device in inches.
`onefile`	should all plots appear in one file or in separate files?
`family`	one of the device-independent font families, `"sans"`, `"serif"` and `"mono"`, or a character string specify a font family to be searched for in a system-dependent way. On unix-alikes (incl.\ Mac), see the 'Cairo fonts' section in the help for `X11`.
`title`	title string to embed as the '`/Title`' field in the file. Defaults to `"R Graphics Output"`.
`fonts`	a character vector specifying R graphics font family names for additional fonts which will be included in the PDF file. Defaults to `NULL`.
`version`	a string describing the PDF version that will be required to view the output. This is a minimum, and will be increased (with a warning) if necessary. Defaults to `"1.4"`, but see 'Details'.
`paper`	the target paper size. The choices are `"a4"`, `"letter"`, `"legal"` (or `"us"`) and `"executive"` (and these can be capitalized), or `"a4r"` and `"USr"` for rotated ('landscape'). The default is `"special"`, which means that the `width` and `height` specify the paper size. A further choice is `"default"`; if this is selected, the papersize is taken from the option `"papersize"` if that is set and as `"a4"` if it is unset or empty. Defaults to `"special"`.
`encoding`	the name of an encoding file. See `postscript` for details. Defaults to `"default"`.
`bg`	the initial background colour: can be overridden by setting par("bg").
`fg`	the initial foreground color to be used. Defaults to `"black"`.
`pointsize`	the default pointsize of plotted text (in big points).
`pagecentre`	logical: should the device region be centred on the page? – is only relevant for `paper != "special"`. Defaults to `TRUE`.

`colormodel`	a character string describing the color model: currently allowed values are `"srgb"`, `"gray"` (or `"grey"`) and `"cmyk"`. Defaults to `"srgb"`. See section 'Color models'.
`useDingbats`	logical. Should small circles be rendered *via* the Dingbats font? Defaults to `FALSE`. If `TRUE`, this can produce smaller and better output, but there can font display problems in broken PDF viewers: although this font is one of the 14 guaranteed to be available in all PDF viewers, that guarantee is not always honoured. For Unix-alikes (including macOS) see the 'Note' for a possible fix for some viewers.
`useKerning`	logical. Should kerning corrections be included in setting text and calculating string widths? Defaults to `TRUE`.
`fillOddEven`	logical controlling the polygon fill mode: see `polygon` for details. Defaults to `FALSE`.
`compress`	logical. Should PDF streams be generated with Flate compression? Defaults to `TRUE`.
`horizontal`	the orientation of the printed image, a logical. Defaults to true, that is landscape orientation on paper sizes with width less than height.
`print.it`	logical: should the file be printed when the device is closed? (This only applies if `file` is a real file name.) Defaults to false.
`command`	the command to be used for 'printing'. Defaults to `"default"`, the value of option `"printcmd"`. The length limit is `2*PATH_MAX`, typically 8096 bytes on unix systems and 520 bytes on windows.
`antialias`	string, the type of anti-aliasing (if any) to be used; defaults to `"default"`.
`defaultfont`	logical: should the device use xfig's default font?
`textspecial`	logical: should the device set the textspecial flag for all text elements. This is useful when generating pstex from xfig figures.

Value

`[any]`
The results of the evaluation of the code argument.

Functions

- `with_bmp()`: BMP device
- `with_cairo_pdf()`: CAIRO_PDF device
- `with_cairo_ps()`: CAIRO_PS device
- `with_pdf()`: PDF device
- `with_postscript()`: POSTSCRIPT device
- `with_svg()`: SVG device
- `with_tiff()`: TIFF device
- `with_xfig()`: XFIG device
- `with_png()`: PNG device
- `with_jpeg()`: JPEG device

See Also

withr for examples

Devices

Examples

```
# dimensions are in inches
with_pdf(file.path(tempdir(), "test.pdf"), width = 7, height = 5,
  plot(runif(5))
)

# dimensions are in pixels
with_png(file.path(tempdir(), "test.png"), width = 800, height = 600,
  plot(runif(5))
)
```

---

withr *Execute code in temporarily altered environment*

Description

All functions prefixed by `with_` work as follows. First, a particular aspect of the global environment is modified (see below for a list). Then, custom code (passed via the `code` argument) is executed. Upon completion or error, the global environment is restored to the previous state. Each `with_` function has a `local_` variant, which instead resets the state when the current evaluation context ends (such as the end of a function).

Arguments pattern

`new`	`[various]`	Values for setting
`code`	`[any]`	Code to execute in the temporary environment
`...`		Further arguments

Usage pattern

`with_...(new, code, ...)`

withr functions

- `with_collate()`: collation order
- `with_dir()`: working directory
- `with_envvar()`: environment variables
- `with_libpaths()`: library paths, replacing current libpaths

- `with_locale()`: any locale setting
- `with_makevars()`: Makevars variables
- `with_options()`: options
- `with_par()`: graphics parameters
- `with_path()`: `PATH` environment variable
- `with_sink()`: output redirection

Creating new "with" functions

All `with_` functions are created by a helper function, `with_()`. This functions accepts two arguments: a setter function and an optional resetter function. The setter function is expected to change the global state and return an "undo instruction". This undo instruction is then passed to the resetter function, which changes back the global state. In many cases, the setter function can be used naturally as resetter.

Author(s)

Maintainer: Lionel Henry `<lionel@posit.co>`

Authors:

- Jim Hester
- Kirill Müller `<krlmlr+r@mailbox.org>`
- Kevin Ushey `<kevinushey@gmail.com>`
- Hadley Wickham `<hadley@posit.co>`
- Winston Chang

Other contributors:

- Jennifer Bryan [contributor]
- Richard Cotton [contributor]
- Posit Software, PBC [copyright holder, funder]

See Also

Useful links:

- `https://withr.r-lib.org`
- `https://github.com/r-lib/withr#readme`
- Report bugs at `https://github.com/r-lib/withr/issues`

Examples

```
getwd()
with_dir(tempdir(), getwd())
getwd()

Sys.getenv("WITHR")
with_envvar(c("WITHR" = 2), Sys.getenv("WITHR"))
Sys.getenv("WITHR")

with_envvar(c("A" = 1),
  with_envvar(c("A" = 2), action = "suffix", Sys.getenv("A"))
)

# local variants are best used within other functions
f <- function(x) {
  local_envvar(c("WITHR" = 2))
  Sys.getenv("WITHR")
}
Sys.getenv("WITHR")
```

`with_collate` *Collation Order*

Description

Temporarily change collation order by changing the value of the `LC_COLLATE` locale.

Usage

```
with_collate(new, code)

local_collate(new = list(), .local_envir = parent.frame())
```

Arguments

`new`	`[character(1)]` New collation order
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
# Modify collation order:
x <- c("bernard", "bérénice", "béatrice", "boris")
with_collate("fr_FR", sort(x))
with_collate("C", sort(x))
```

`with_connection` *Connections which close themselves*

Description

R file connections which are automatically closed.

Usage

```
with_connection(con, code)

local_connection(con, .local_envir = parent.frame())
```

Arguments

`con`	For `with_connection()` a named list with the connection(s) to create. For `local_connection()` the code to create a single connection, which is then returned.
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
with_connection(list(con = file("foo", "w")), {
  writeLines(c("foo", "bar"), con)
})

read_foo <- function() {
  readLines(local_connection(file("foo", "r")))
}
read_foo()

unlink("foo")
```

`with_db_connection` *DBMS Connections which disconnect themselves.*

Description

Connections to Database Management Systems which automatically disconnect. In particular connections which are created with `DBI::dbConnect()` and closed with `DBI::dbDisconnect()`.

Usage

```
with_db_connection(con, code)

local_db_connection(con, .local_envir = parent.frame())
```

Arguments

`con`	For `with_db_connection()` a named list with the connection(s) to create. For `local_db_connection()` the code to create a single connection, which is then returned.
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
db <- tempfile()
with_db_connection(
  list(con = DBI::dbConnect(RSQLite::SQLite(), db)), {
    DBI::dbWriteTable(con, "mtcars", mtcars)
})

head_db_table <- function(...) {
  con <- local_db_connection(DBI::dbConnect(RSQLite::SQLite(), db))
  head(DBI::dbReadTable(con, "mtcars"), ...)
}
head_db_table()
unlink(db)
```

`with_dir` *Working directory*

Description

Temporarily change the current working directory.

Usage

```
with_dir(new, code)

local_dir(new = list(), .local_envir = parent.frame())
```

Arguments

`new`	`[character(1)]` New working directory
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`setwd()`

Examples

```
getwd()

with_dir(tempdir(), getwd())
```

`with_envvar` *Environment variables*

Description

Temporarily change system environment variables.

Usage

```
with_envvar(new, code, action = "replace")

local_envvar(
  .new = list(),
  ...,
  action = "replace",
  .local_envir = parent.frame()
)
```

Arguments

`new, .new`	`[named character]` New environment variables
`code`	`[any]` Code to execute in the temporary environment
`action`	should new values `"replace"`, `"prefix"` or `"suffix"` existing variables with the same name.
`...`	Named arguments with new environment variables.
`.local_envir`	`[environment]` The environment to use for scoping.

Details

if `NA` is used those environment variables will be unset. If there are any duplicated variable names only the last one is used.

Value

`[any]`
The results of the evaluation of the `code` argument.

See Also

`withr` for examples

`Sys.setenv()`

Examples

```
with_envvar(new = c("GITHUB_PAT" = "abcdef"), Sys.getenv("GITHUB_PAT"))

# with_envvar unsets variables after usage
Sys.getenv("TEMP_SECRET")
with_envvar(new = c("TEMP_SECRET" = "secret"), Sys.getenv("TEMP_SECRET"))
Sys.getenv("TEMP_SECRET")
```

`with_file` *Files which delete themselves*

Description

Create files, which are then automatically removed afterwards.

Usage

```
with_file(file, code)

local_file(.file, ..., .local_envir = parent.frame())
```

Arguments

`file, .file`	`[named list]` Files to create.
`code`	`[any]` Code to execute in the temporary environment
`...`	Additional (possibly named) arguments of files to create.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
with_file("file1", {
  writeLines("foo", "file1")
  readLines("file1")
})

with_file(list("file1" = writeLines("foo", "file1")), {
  readLines("file1")
})
```

---

with_gctorture2 *Torture Garbage Collector*

Description

Temporarily turn gctorture2 on.

Usage

```
with_gctorture2(new, code, wait = new, inhibit_release = FALSE)
```

Arguments

new	[integer] run GC every 'step' allocations.
code	[any] Code to execute in the temporary environment
wait	integer; number of allocations to wait before starting GC torture.
inhibit_release	logical; do not release free objects for re-use: use with caution.

Value

[any]
The results of the evaluation of the code argument.

See Also

withr for examples

`with_language` *Language*

Description

Temporarily change the language used for translations.

Usage

```
with_language(lang, code)

local_language(lang, .local_envir = parent.frame())
```

Arguments

`lang`	A BCP47 language code like "en" (English), "fr" (French), "fr_CA" (French Canadian). Formally, this is a lower case two letter ISO 639 country code, optionally followed by "_" or "-" and an upper case two letter ISO 3166 region code.
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	`[environment]` The environment to use for scoping.

Examples

```
with_language("en", try(mean[[1]]))
with_language("fr", try(mean[[1]]))
with_language("es", try(mean[[1]]))
```

`with_libpaths` *Library paths*

Description

Temporarily change library paths.

Usage

```
with_libpaths(new, code, action = "replace")

local_libpaths(new = list(), action = "replace", .local_envir = parent.frame())
```

Arguments

`new`	`[character]` New library paths
`code`	`[any]` Code to execute in the temporary environment
`action`	`[character(1)]` should new values `"replace"`, `"prefix"` or `"suffix"` existing paths.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`.libPaths()`

Other libpaths: `with_temp_libpaths()`

Examples

```
.libPaths()
new_lib <- tempfile()
dir.create(new_lib)
with_libpaths(new_lib, print(.libPaths()))
unlink(new_lib, recursive = TRUE)
```

`with_locale` *Locale settings*

Description

Temporarily change locale settings.

Usage

```
with_locale(new, code)

local_locale(.new = list(), ..., .local_envir = parent.frame())
```

Arguments

`new, .new`	`[named character]` New locale settings
`code`	`[any]` Code to execute in the temporary environment
`...`	Additional arguments with locale settings.
`.local_envir`	`[environment]` The environment to use for scoping.

Details

Setting the `LC_ALL` category is currently not implemented.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`Sys.setlocale()`

Examples

```
## Change locale for time:
df <- data.frame(
  stringsAsFactors = FALSE,
  date = as.Date(c("2019-01-01", "2019-02-01")),
  value = c(1, 2)
)
with_locale(new = c("LC_TIME" = "es_ES"), code = plot(df$date, df$value))
## Compare with:
#  plot(df$date, df$value)

## Month names:
with_locale(new = c("LC_TIME" = "en_GB"), format(ISOdate(2000, 1:12, 1), "%B"))
with_locale(new = c("LC_TIME" = "es_ES"), format(ISOdate(2000, 1:12, 1), "%B"))

## Change locale for currencies:
with_locale(new = c("LC_MONETARY" = "it_IT"), Sys.localeconv())
with_locale(new = c("LC_MONETARY" = "en_US"), Sys.localeconv())

## Ordering:
x <- c("bernard", "bérénice", "béatrice", "boris")
with_locale(c(LC_COLLATE = "fr_FR"), sort(x))
with_locale(c(LC_COLLATE = "C"), sort(x))
```

with_makevars *Makevars variables*

Description

Temporarily change contents of an existing `Makevars` file.

Usage

```
with_makevars(
  new,
  code,
  path = makevars_user(),
  assignment = c("=", ":=", "?=", "+=")
)

local_makevars(
  .new = list(),
  ...,
  .path = makevars_user(),
  .assignment = c("=", ":=", "?=", "+="),
  .local_envir = parent.frame()
)
```

Arguments

`new, .new`	`[named character]` New variables and their values
`code`	`[any]` Code to execute in the temporary environment
`path, .path`	`[character(1)]` location of existing `Makevars` file to modify.
`assignment, .assignment`	`[character(1)]` assignment type to use.
`...`	Additional new variables and their values.
`.local_envir`	`[environment]` The environment to use for scoping.

Details

If no `Makevars` file exists or the fields in `new` do not exist in the existing `Makevars` file then the fields are added to the new file. Existing fields which are not included in `new` are appended unchanged. Fields which exist in `Makevars` and in `new` are modified to use the value in `new`.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
writeLines("void foo(int* bar) { *bar = 1; }\n", "foo.c")
system("R CMD SHLIB --preclean -c foo.c")
with_makevars(c(CFLAGS = "-O3"), system("R CMD SHLIB --preclean -c foo.c"))
unlink(c("foo.c", "foo.so"))
```

`with_options` *Options*

Description

Temporarily change global options.

Usage

```
with_options(new, code)

local_options(.new = list(), ..., .local_envir = parent.frame())
```

Arguments

`new, .new`	`[named list]` New options and their values
`code`	`[any]` Code to execute in the temporary environment
`...`	Additional options and their values
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`options()`

Examples

```
# number of significant digits to print
getOption("digits")
# modify temporarily the number of significant digits to print
with_options(list(digits = 3), getOption("digits"))
with_options(list(digits = 3), print(pi))

# modify temporarily the character to be used as the decimal point
getOption("digits")
with_options(list(OutDec = ","), print(pi))

# modify temporarily multiple options
with_options(list(OutDec = ",", digits = 3), print(pi))

# modify, within the scope of the function, the number of
# significant digits to print
print_3_digits <- function(x) {
  # assign 3 to the option "digits" for the rest of this function
  # after the function exits, the option will return to its previous
  # value
  local_options(list(digits = 3))
  print(x)
}

print_3_digits(pi)  # returns 3.14
print(pi)           # returns 3.141593
```

---

with_package *Execute code with a modified search path*

Description

`with_package()` attaches a package to the search path, executes the code, then removes the package from the search path. The package namespace is *not* unloaded however. `with_namespace()` does the same thing, but attaches the package namespace to the search path, so all objects (even unexported ones) are also available on the search path.

Usage

```
with_package(
  package,
  code,
  pos = 2,
  lib.loc = NULL,
  character.only = TRUE,
  logical.return = FALSE,
  warn.conflicts = FALSE,
  quietly = TRUE,
  verbose = getOption("verbose")
)

local_package(
  package,
  pos = 2,
  lib.loc = NULL,
  character.only = TRUE,
  logical.return = FALSE,
  warn.conflicts = FALSE,
  quietly = TRUE,
  verbose = getOption("verbose"),
  .local_envir = parent.frame()
)

with_namespace(package, code, warn.conflicts = FALSE)

local_namespace(package, .local_envir = parent.frame(), warn.conflicts = FALSE)

with_environment(
  env,
  code,
  pos = 2L,
  name = format(env),
  warn.conflicts = FALSE
)

local_environment(
  env,
  pos = 2L,
  name = format(env),
  warn.conflicts = FALSE,
  .local_envir = parent.frame()
)
```

Arguments

`package`	`[character(1)]` package name to load.
`code`	`[any]` Code to execute in the temporary environment
`pos`	the position on the search list at which to attach the loaded namespace. Can also be the name of a position on the current search list as given by `search()`.
`lib.loc`	a character vector describing the location of R library trees to search through, or `NULL`. The default value of `NULL` corresponds to all libraries currently known to `.libPaths()`. Non-existent library trees are silently ignored.
`character.only`	a logical indicating whether `package` or `help` can be assumed to be character strings.

`logical.return`	logical. If it is `TRUE`, `FALSE` or `TRUE` is returned to indicate success.
`warn.conflicts`	logical. If `TRUE`, warnings are printed about `conflicts` from attaching the new package. A conflict is a function masking a function, or a non-function masking a non-function. The default is `TRUE` unless specified as `FALSE` in the `conflicts.policy` option.
`quietly`	a logical. If `TRUE`, no message confirming package attaching is printed, and most often, no errors/warnings are printed if package attaching fails.
`verbose`	a logical. If `TRUE`, additional diagnostics are printed.
`.local_envir`	`[environment]` The environment to use for scoping.
`env`	`[environment()]` Environment to attach.
`name`	name to use for the attached database. Names starting with `package:` are reserved for `library`.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
## Not run:
with_package("ggplot2", {
  ggplot(mtcars) + geom_point(aes(wt, hp))
})

## End(Not run)
```

`with_par` *Graphics parameters*

Description

Temporarily change graphics parameters.

Usage

```
with_par(new, code, no.readonly = FALSE)

local_par(
  .new = list(),
  ...,
  no.readonly = FALSE,
  .local_envir = parent.frame()
)
```

Arguments

`new, .new`	`[named list]` New graphics parameters and their values
`code`	`[any]` Code to execute in the temporary environment
`no.readonly`	`[logical(1)]` see `par()` documentation.
`...`	Additional graphics parameters and their values.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`par()`

Examples

```
old <- par("col" = "black")

# This will be in red
with_par(list(col = "red", pch = 19),
  plot(mtcars$hp, mtcars$wt)
)

# This will still be in black
plot(mtcars$hp, mtcars$wt)

par(old)
```

with_path — *PATH environment variable*

Description

Temporarily change the system search path.

Usage

```
with_path(new, code, action = c("prefix", "suffix", "replace"))

local_path(
  new = list(),
  action = c("prefix", "suffix", "replace"),
  .local_envir = parent.frame()
)
```

Arguments

`new`	`[character]` New `PATH` entries
`code`	`[any]` Code to execute in the temporary environment
`action`	`[character(1)]` Should new values `"replace"`, `"prefix"` (the default) or `"suffix"` existing paths
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`Sys.setenv()`

Examples

```
# temporarily modify the system PATH, *prefixing* the current path
with_path(getwd(), Sys.getenv("PATH"))
# temporarily modify the system PATH, *appending* to the current path
with_path(getwd(), Sys.getenv("PATH"), "suffix")
```

`with_rng_version` *RNG version*

Description

Change the RNG version and restore it afterwards.

Usage

```
with_rng_version(version, code)

local_rng_version(version, .local_envir = parent.frame())
```

Arguments

`version`	`[character(1)]` an R version number, e.g. `"3.5.0"`, to switch to the RNG this version of R uses. See `RNGversion()`.
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	The environment to apply the change to.

Details

`with_rng_version()` runs the code with the specified RNG version and resets it afterwards.

`local_rng_version()` changes the RNG version for the caller execution environment.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`RNGversion()`, `RNGkind()`, `with_seed()`.

Examples

```
RNGkind()
with_rng_version("3.0.0", RNGkind())
with_rng_version("1.6.0", RNGkind())

with_rng_version("3.0.0",
  with_seed(42, sample(1:100, 3)))

with_rng_version("1.6.0",
  with_seed(42, sample(1:100, 3)))
```

```
RNGkind()

fun1 <- function() {
  local_rng_version("3.0.0")
  with_seed(42, sample(1:100, 3))
}

fun2 <- function() {
  local_rng_version("1.6.0")
  with_seed(42, sample(1:100, 3))
}

RNGkind()
fun1()
fun2()
RNGkind()
```

with_seed *Random seed*

Description

with_seed() runs code with a specific random seed and resets it afterwards.

with_preserve_seed() runs code with the current random seed and resets it afterwards.

Usage

```
with_seed(
  seed,
  code,
  .rng_kind = NULL,
  .rng_normal_kind = NULL,
  .rng_sample_kind = NULL
)

local_seed(
  seed,
  .local_envir = parent.frame(),
  .rng_kind = NULL,
  .rng_normal_kind = NULL,
  .rng_sample_kind = NULL
)

with_preserve_seed(code)

local_preserve_seed(.local_envir = parent.frame())
```

Arguments

`seed`	`[integer(1)]` The random seed to use to evaluate the code.
`code`	`[any]` Code to execute in the temporary environment
`.rng_kind`, `.rng_normal_kind`, `.rng_sample_kind`	`[character(1)]` Kind of RNG to use. Passed as the `kind`, `normal.kind`, and `sample.kind` arguments of `RNGkind()`.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

Examples

```
# Same random values:
with_preserve_seed(runif(5))
with_preserve_seed(runif(5))

# Use a pseudorandom value as seed to advance the RNG and pick a different
# value for the next call:
with_seed(seed <- sample.int(.Machine$integer.max, 1L), runif(5))
with_seed(seed, runif(5))
with_seed(seed <- sample.int(.Machine$integer.max, 1L), runif(5))
```

`with_sink` *Output redirection*

Description

Temporarily divert output to a file via `sink()`. For sinks of type `message`, an error is raised if such a sink is already active.

Usage

```
with_output_sink(new, code, append = FALSE, split = FALSE)

local_output_sink(
  new = list(),
  append = FALSE,
  split = FALSE,
  .local_envir = parent.frame()
)

with_message_sink(new, code, append = FALSE)

local_message_sink(new = list(), append = FALSE, .local_envir = parent.frame())
```

Arguments

`new`	`[character(1)|connection]` A writable connection or a character string naming the file to write to. Passing `NULL` will throw an error.
`code`	`[any]` Code to execute in the temporary environment
`append`	logical. If `TRUE`, output will be appended to `file`; otherwise, it will overwrite the contents of `file`.
`split`	logical: if `TRUE`, output will be sent to the new sink and to the current output stream, like the Unix program `tee`.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`sink()`

`with_tempfile` *Temporary files and directories*

Description

Temporarily create a file or directory, which will automatically deleted once you're finished with it.

Usage

```
with_tempfile(
  new,
  code,
  envir = parent.frame(),
  .local_envir = parent.frame(),
  pattern = "file",
  tmpdir = tempdir(),
  fileext = ""
)

local_tempfile(
  new = NULL,
  lines = NULL,
  envir = parent.frame(),
  .local_envir = parent.frame(),
  pattern = "file",
  tmpdir = tempdir(),
  fileext = ""
)

with_tempdir(
  code,
  clean = TRUE,
  pattern = "file",
  tmpdir = tempdir(),
  fileext = ""
)

local_tempdir(
  pattern = "file",
  tmpdir = tempdir(),
  fileext = "",
  .local_envir = parent.frame(),
  clean = TRUE
)
```

Arguments

`new`	`[character vector]` (Deprecated for `local_tempfile()`) Names of temporary file handles to create.
`code`	`[any]` Code to execute in the temporary environment
`envir`	`[environment]` Deprecated in favor of `.local_envir`.
`.local_envir`	`[environment]` The environment to use for scoping.
`pattern`	a non-empty character vector giving the initial part of the name.
`tmpdir`	a non-empty character vector giving the directory name
`fileext`	a non-empty character vector giving the file extension
`lines`	Optionally, supply a character vector of lines to be written to `path`. This is useful if you want to seed the file with some default content.

clean	[logical(1)] A logical indicating if the temporary directory should be deleted after use (TRUE, default) or left alone (FALSE).

Value

[any]
The results of the evaluation of the code argument.

See Also

withr for examples

Examples

```
# local_tempfile() is the easiest to use because it returns a path
local({
  path1 <<- local_tempfile(lines = c("x,y", "1,2"))
  readLines(path1)
})
# the file is deleted automatically
file.exists(path1)

# with_tempfile() is a bit trickier; the first argument gives the name
# of a variable that will contain the path:
with_tempfile("path2", {
  print(path2)
  write.csv(iris, path2)
  file.size(path2)
})

# Note that this variable is only available in the scope of with_tempfile
try(path2)
```

`with_temp_libpaths` *Library paths*

Description

Temporarily prepend a new temporary directory to the library paths.

Usage

```
with_temp_libpaths(code, action = "prefix")

local_temp_libpaths(action = "prefix", .local_envir = parent.frame())
```

Arguments

`code`	`[any]` Code to execute in the temporary environment
`action`	`[character(1)]` should new values `"replace"`, `"prefix"` or `"suffix"` existing paths.
`.local_envir`	`[environment]` The environment to use for scoping.

Value

`[any]`
The results of the evaluation of the `code` argument.

See Also

`withr` for examples

`.libPaths()`

Other libpaths: `with_libpaths()`

`with_timezone` *Time zone*

Description

Change the time zone, and restore it afterwards.

Usage

```
with_timezone(tz, code)

local_timezone(tz, .local_envir = parent.frame())
```

Arguments

`tz`	`[character(1)]` a valid time zone specification, note that time zone names might be platform dependent.
`code`	`[any]` Code to execute in the temporary environment
`.local_envir`	The environment to apply the change to.

Details

`with_timezone()` runs the code with the specified time zone and resets it afterwards.

`local_timezone()` changes the time zone for the caller execution environment.

Value

`[any]`
The results of the evaluation of the code argument.

See Also

`withr` for examples

`Sys.timezone()`.

Examples

```
Sys.time()
with_timezone("Europe/Paris", print(Sys.time()))
with_timezone("America/Los_Angeles", print(Sys.time()))

fun1 <- function() {
  local_timezone("CET")
  print(Sys.time())
}

fun2 <- function() {
  local_timezone("America/Los_Angeles")
  print(Sys.time())
}
Sys.time()
fun1()
fun2()
Sys.time()
```

Index

∗ **libpaths**
    with_libpaths, 19
    with_temp_libpaths, 34
∗ **local-related functions**
    defer, 2
.libPaths, *25*
.libPaths(), *20*, *35*

conflicts, *26*
connection, *32*

defer, 2
defer_parent *(*defer*)*, 2
deferred_clear *(*defer*)*, 2
deferred_run *(*defer*)*, 2
Devices, *10*
devices, 4

library, *26*
local_bmp *(*devices*)*, 4
local_cairo_pdf *(*devices*)*, 4
local_cairo_ps *(*devices*)*, 4
local_collate *(*with_collate*)*, 12
local_connection *(*with_connection*)*, 13
local_db_connection
    *(*with_db_connection*)*, 14
local_dir *(*with_dir*)*, 15
local_environment *(*with_package*)*, 24
local_envvar *(*with_envvar*)*, 16
local_file *(*with_file*)*, 17
local_jpeg *(*devices*)*, 4
local_language *(*with_language*)*, 19
local_libpaths *(*with_libpaths*)*, 19
local_locale *(*with_locale*)*, 20
local_makevars *(*with_makevars*)*, 22
local_message_sink *(*with_sink*)*, 31
local_namespace *(*with_package*)*, 24
local_options *(*with_options*)*, 23
local_output_sink *(*with_sink*)*, 31
local_package *(*with_package*)*, 24
local_par *(*with_par*)*, 26
local_path *(*with_path*)*, 28
local_pdf *(*devices*)*, 4
local_png *(*devices*)*, 4
local_postscript *(*devices*)*, 4
local_preserve_seed *(*with_seed*)*, 30
local_rng_version *(*with_rng_version*)*, 29
local_seed *(*with_seed*)*, 30
local_svg *(*devices*)*, 4
local_temp_libpaths
    *(*with_temp_libpaths*)*, 34
local_tempdir *(*with_tempfile*)*, 32
local_tempfile *(*with_tempfile*)*, 32
local_tiff *(*devices*)*, 4
local_timezone *(*with_timezone*)*, 35
local_xfig *(*devices*)*, 4

on.exit(), *2*
options(), *23*

par(), *27*
parent.frame(), *3*
polygon, *9*
postscript, *8*

RNGkind(), *29*, *31*
RNGversion(), *29*

search, *25*
setwd(), *15*
sink(), *31*, *32*
Sys.setenv(), *17*, *28*
Sys.setlocale(), *21*
Sys.timezone(), *36*

with_(), *11*
with_bmp *(*devices*)*, 4
with_cairo_pdf *(*devices*)*, 4
with_cairo_ps *(*devices*)*, 4
with_collate, 12
with_collate(), *10*

with_connection, 13
with_db_connection, 14
with_dev *(*devices*)*, 4
with_device *(*devices*)*, 4
with_dir, 15
with_dir(), *10*
with_environment *(*with_package*)*, 24
with_envvar, 16
with_envvar(), *10*
with_file, 17
with_gctorture2, 18
with_jpeg *(*devices*)*, 4
with_language, 19
with_libpaths, 19, *35*
with_libpaths(), *10*
with_locale, 20
with_locale(), *11*
with_makevars, 22
with_makevars(), *11*
with_message_sink *(*with_sink*)*, 31
with_namespace *(*with_package*)*, 24
with_options, 23
with_options(), *11*
with_output_sink *(*with_sink*)*, 31
with_package, 24
with_par, 26
with_par(), *11*
with_path, 28
with_path(), *11*
with_pdf *(*devices*)*, 4
with_png *(*devices*)*, 4
with_postscript *(*devices*)*, 4
with_preserve_seed *(*with_seed*)*, 30
with_rng_version, 29
with_seed, 30
with_seed(), *29*
with_sink, 31
with_sink(), *11*
with_svg *(*devices*)*, 4
with_temp_libpaths, *20*, 34
with_tempdir *(*with_tempfile*)*, 32
with_tempfile, 32
with_tiff *(*devices*)*, 4
with_timezone, 35
with_xfig *(*devices*)*, 4
withr, *10*, 10, *13–15*, *17*, *18*, *20*, *21*, *23*, *26–29*, *31*, *32*, *34–36*
withr-package *(*withr*)*, 10

X11, *8*