Package 'withr'

July 29, 2017

Encoding UTF-8

Title Run Code 'With' Temporarily Modified Global State

Version 2.0.0

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.

URL http://github.com/r-lib/withr#readme

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

Depends R (>= 3.0.2)

License GPL (>= 2)

LazyData true

Imports stats, graphics

Suggests testthat

Collate 'local_.R' 'with_.R' 'collate.R' 'defer.R' 'dir.R' 'env.R' 'libpaths.R' 'locale.R' 'makevars.R' 'options.R' 'par.R' 'path.R' 'seed.R' 'wrap.R' 'sink.R' 'utils.R' 'with.R'

RoxygenNote 6.0.1

NeedsCompilation no

Author Jim Hester [aut, cre],
Kirill M<U+00FC>ller [aut],
Kevin Ushey [aut],
Hadley Wickham [aut],
Winston Chang [aut],
RStudio [cph]

Maintainer Jim Hester <james.f.hester@gmail.com>

Repository CRAN

Date/Publication 2017-07-28 22:56:28 UTC

R topics documented:

`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"))
```

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.

Author(s)

Kevin Ushey

Examples

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

# create tempfile path
path <- tempfile()

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

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

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

`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.

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.

Examples

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

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

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

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, code, .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

with_dir *Working directory*

Description

Temporarily change the current working directory.

Usage

```
with_dir(new, code)

local_dir(new, code, .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()

---

with_envvar *Environment variables*

Description

Temporarily change system environment variables.

Usage

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

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

Arguments

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.
.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()

---

with_libpaths *Library paths*

Description

Temporarily change library paths.

Usage

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

local_libpaths(new, code, 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

`with_locale` *Locale settings*

Description

Temporarily change locale settings.

Usage

```
with_locale(new, code)

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

Arguments

`new`	`[named character]` New locale settings
`code`	`[any]` Code to execute in the temporary environment
`.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()`

`with_makevars` *Makevars variables*

Description

Temporarily change contents of an existing `Makevars` file.

Usage

```
with_makevars(new, code, path = file.path("~", ".R", "Makevars"),
  assignment = c("=", ":=", "?=", "+="))
```

Arguments

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

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

with_options *Options*

Description

Temporarily change global options.

Usage

```
with_options(new, code)

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

Arguments

new	[named list] New options and their values
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

options()

with_par *Graphics parameters*

Description

Temporarily change graphics parameters.

Usage

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

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

Arguments

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.
.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()

---

with_path *PATH environment variable*

Description

Temporarily change the system search path.

Usage

```
with_path(new, code, action = "prefix")

local_path(new, code, action = "prefix", .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" 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()

---

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)

with_preserve_seed(code)
```

Arguments

seed	[integer(1)] The random seed to use to evaluate the code.
code	[any] Code to execute in the temporary environment

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, code, append = FALSE, split = FALSE,
  .local_envir = parent.frame())

with_message_sink(new, code, append = FALSE)

local_message_sink(new, code, 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_temp_libpaths *Library paths*

Description

Temporarily prepend a new temporary directory to the library paths.

Usage

```
with_temp_libpaths(code)

local_temp_libpaths(code, .local_envir = parent.frame())
```

Arguments

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

.libPaths()

Other libpaths: with_libpaths

Index