Package: callr (via r-universe)

July 5, 2024

Title Call R from R

Version 3.7.6.9000

Description It is sometimes useful to perform a computation in a separate R process, without affecting the current R process at all. This packages does exactly that.

License MIT + file LICENSE

URL `https://callr.r-lib.org`, `https://github.com/r-lib/callr`

BugReports `https://github.com/r-lib/callr/issues`

Depends R (>= 3.4)

Imports processx (>= 3.6.1), R6, utils

Suggests asciicast (>= 2.3.1), cli (>= 1.1.0), mockery, ps, rprojroot, spelling, testthat (>= 3.2.0), withr (>= 2.3.0)

Config/Needs/website r-lib/asciicast, glue, htmlwidgets, igraph, tibble, tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1.9000

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

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

RemoteRef HEAD

RemoteSha cc2d8884fb01e736f319a74fbf8449ac11ede9ed

Contents

add_hook *Add a user hook to be executed before launching an R subprocess*

Description

This function allows users of `callr` to specify functions that get invoked whenever an R session is launched. The function can modify the environment variables and command line arguments.

Usage

```
add_hook(...)
```

Arguments

`...`	Named argument specifying a hook function to add, or `NULL` to delete the named hook.

Details

The prototype of the hook function is `function (options)`, and it is expected to return the modified `options`.

Value

`add_hook` is called for its side-effects.

`default_repos` *Default value for the* `repos` *option in callr subprocesses*

Description

callr sets the `repos` option in subprocesses, to make sure that a CRAN mirror is set up. This is because the subprocess cannot bring up the menu of CRAN mirrors for the user to choose from.

Usage

```
default_repos()
```

Value

Named character vector, the default value of the `repos` option in callr subprocesses.

Examples

```
default_repos()
```

`r` *Evaluate an expression in another R session*

Description

From `callr` version 2.0.0, `r()` is equivalent to `r_safe()`, and tries to set up a less error prone execution environment. In particular:

- Ensures that at least one reasonable CRAN mirror is set up.
- Adds some command line arguments to avoid saving `.RData` files, etc.
- Ignores the system and user profiles (by default).
- Sets various environment variables: `CYGWIN` to avoid warnings about DOS-style paths, `R_TESTS` to avoid issues when `callr` is invoked from unit tests, `R_BROWSER` and `R_PDFVIEWER` to avoid starting a browser or a PDF viewer. See `rcmd_safe_env()`.

Usage

```
r(
  func,
  args = list(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = NULL,
  stderr = NULL,
  poll_connection = TRUE,
  error = getOption("callr.error", "error"),
  cmdargs = c("--slave", "--no-save", "--no-restore"),
  show = FALSE,
  callback = NULL,
  block_callback = NULL,
  spinner = show && interactive(),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  timeout = Inf,
  package = FALSE,
  arch = "same",
  ...
)

r_safe(
  func,
  args = list(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = NULL,
  stderr = NULL,
  poll_connection = TRUE,
  error = getOption("callr.error", "error"),
  cmdargs = c("--slave", "--no-save", "--no-restore"),
  show = FALSE,
  callback = NULL,
  block_callback = NULL,
  spinner = show && interactive(),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  timeout = Inf,
  package = FALSE,
  arch = "same",
  ...
)
```

Arguments

`func`	Function object to call in the new R process. The function should be self-contained and only refer to other functions and use variables explicitly from other packages using the `::` notation. By default the environment of the function is set to `.GlobalEnv` before passing it to the child process. (See the `package` option if you want to keep the environment.) Because of this, it is good practice to create an anonymous function and pass that to `callr`, instead of passing a function object from a (base or other) package. In particular `r(.libPaths)`

does not work, because `.libPaths` is defined in a special environment, but

```
r(function() .libPaths())
```

works just fine.

`args`	Arguments to pass to the function. Must be a list.
`libpath`	The library path.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`stdout`	The name of the file the standard output of the child R process will be written to. If the child process runs with the `--slave` option (the default), then the commands are not echoed and will not be shown in the standard output. Also note that you need to call `print()` explicitly to show the output of the command(s). IF `NULL`, then standard output is not returned, but it is recorded and included in the error object if an error happens. Various special values for this argument such as `"|"` are explained in the `stdout` argument of processx::process.
`stderr`	The name of the file the standard error of the child R process will be written to. In particular `message()` sends output to the standard error. If nothing was sent to the standard error, then this file will be empty. This argument can be the same file as `stdout`, in which case they will be correctly interleaved. If this is the string `"2>&1"`, then standard error is redirected to standard output. IF `NULL`, then standard output is not returned, but it is recorded and included in the error object if an error happens. Various special values for this argument such as `"|"` are explained in the `stdout` argument of processx::process.
`poll_connection`	Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the main process.
`error`	What to do if the remote process throws an error. See details below.
`cmdargs`	Command line arguments to pass to the R process. Note that `c("-f", rscript)` is appended to this, `rscript` is the name of the script file to run. This contains a call to the supplied function and some error handling code.
`show`	Logical, whether to show the standard output on the screen while the child process is running. Note that this is independent of the `stdout` and `stderr` arguments. The standard error is not shown currently.
`callback`	A function to call for each line of the standard output and standard error from the child process. It works together with the `show` option; i.e. if `show = TRUE`, and a callback is provided, then the output is shown of the screen, and the callback is also called.
`block_callback`	A function to call for each block of the standard output and standard error. This callback is not line oriented, i.e. multiple lines or half a line can be passed to the callback.
`spinner`	Whether to show a calming spinner on the screen while the child R session is running. By default it is shown if `show = TRUE` and the R session is interactive.
`system_profile`	Whether to use the system profile file.

`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`timeout`	Timeout for the function call to finish. It can be a base::difftime object, or a real number, meaning seconds. If the process does not finish before the timeout period expires, then a `system_command_timeout_error` error is thrown. `Inf` means no timeout.
`package`	Whether to keep the environment of `func` when passing it to the other package. Possible values are: • `FALSE`: reset the environment to `.GlobalEnv`. This is the default. • `TRUE`: keep the environment as is. • `pkg`: set the environment to the `pkg` package namespace.
`arch`	Architecture to use in the child process, for multi-arch builds of R. By default the same as the main process. See `supported_archs()`. If it contains a forward or backward slash character, then it is taken as the path to the R executable. Note that on Windows you need the path to `Rterm.exe`.
`...`	Extra arguments are passed to `processx::run()`.

Details

The `r()` function from before 2.0.0 is called `r_copycat()` now.

Value

Value of the evaluated expression.

Error handling

`callr` handles errors properly. If the child process throws an error, then `callr` throws an error with the same error message in the main process.

The `error` expert argument may be used to specify a different behavior on error. The following values are possible:

- `error` is the default behavior: throw an error in the main process, with a prefix and the same error message as in the subprocess.
- `stack` also throws an error in the main process, but the error is of a special kind, class `callr_error`, and it contains both the original error object, and the call stack of the child, as written out by `utils::dump.frames()`. This is now deprecated, because the error thrown for `"error"` has the same information.
- `debugger` is similar to `stack`, but in addition to returning the complete call stack, it also start up a debugger in the child call stack, via `utils::debugger()`.

The default error behavior can be also set using the `callr.error` option. This is useful to debug code that uses `callr`.

callr uses parent errors, to keep the stacks of the main process and the subprocess(es) in the same error object.

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

Transporting objects

`func` and `args` are copied to the child process by first serializing them into a temporary file using `saveRDS()` and then loading them back into the child session using `readRDS()`. The same strategy is used to copy the result of calling `func(args)` to the main session. Note that some objects, notably those with `externalptr` type, won't work as expected after being saved to a file and loaded back.

For performance reasons `compress=FALSE` is used when serializing with `saveRDS()`, this can be disabled by setting `options(callr.compress_transport = TRUE)`.

See Also

Other callr functions: `r_copycat()`, `r_vanilla()`

Examples

```
# Workspace is empty
r(function() ls())

# library path is the same by default
r(function() .libPaths())
.libPaths()
```

`rcmd` *Run an* `R CMD` *command*

Description

Run an `R CMD` command form within R. This will usually start another R process, from a shell script.

Usage

```
rcmd(
  cmd,
  cmdargs = character(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = NULL,
  stderr = NULL,
  poll_connection = TRUE,
  echo = FALSE,
  show = FALSE,
  callback = NULL,
  block_callback = NULL,
  spinner = show && interactive(),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  timeout = Inf,
  wd = ".",
  fail_on_status = FALSE,
  ...
)

rcmd_safe(
  cmd,
  cmdargs = character(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = NULL,
  stderr = NULL,
  poll_connection = TRUE,
  echo = FALSE,
  show = FALSE,
  callback = NULL,
  block_callback = NULL,
  spinner = show && interactive(),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  timeout = Inf,
  wd = ".",
  fail_on_status = FALSE,
  ...
)
```

Arguments

`cmd`	Command to run. See `R --help` from the command line for the various commands. In the current version of R (3.2.4) these are: `BATCH`, `COMPILE`, `SHLIB`, `INSTALL`, `REMOVE`, `build`, `check`, `LINK`, `Rprof`, `Rdconv`, `Rd2pdf`, `Rd2txt`, `Stangle`, `Sweave`, `Rdiff`, `config`, `javareconf`, `rtags`.
`cmdargs`	Command line arguments.
`libpath`	The library path.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`stdout`	Optionally a file name to send the standard output to.

`stderr`	Optionally a file name to send the standard error to. It may be the same as `stdout`, in which case standard error is redirected to standard output. It can also be the special string `"2>&1"`, in which case standard error will be redirected to standard output.
`poll_connection`	Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the parent.
`echo`	Whether to echo the complete command run by `rcmd`.
`show`	Logical, whether to show the standard output on the screen while the child process is running. Note that this is independent of the `stdout` and `stderr` arguments. The standard error is not shown currently.
`callback`	A function to call for each line of the standard output and standard error from the child process. It works together with the `show` option; i.e. if `show = TRUE`, and a callback is provided, then the output is shown of the screen, and the callback is also called.
`block_callback`	A function to call for each block of the standard output and standard error. This callback is not line oriented, i.e. multiple lines or half a line can be passed to the callback.
`spinner`	Whether to show a calming spinner on the screen while the child R session is running. By default it is shown if `show = TRUE` and the R session is interactive.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`timeout`	Timeout for the function call to finish. It can be a base::difftime object, or a real number, meaning seconds. If the process does not finish before the timeout period expires, then a `system_command_timeout_error` error is thrown. `Inf` means no timeout.
`wd`	Working directory to use for running the command. Defaults to the current working directory.
`fail_on_status`	Whether to throw an R error if the command returns with a non-zero status code. By default no error is thrown.
`...`	Extra arguments are passed to `processx::run()`.

Details

Starting from `callr` 2.0.0, `rcmd()` has safer defaults, the same as the `rcmd_safe()` default values. Use `rcmd_copycat()` for the old defaults.

Value

A list with the command line `$command`), standard output (`$stdout`), standard error (`stderr`), exit status (`$status`) of the external `R CMD` command, and whether a timeout was reached (`$timeout`).

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

See Also

Other R CMD commands: `rcmd_bg()`, `rcmd_copycat()`

Examples

```
rcmd("config", "CC")
```

---

rcmd_bg *Run an `R CMD` command in the background*

Description

The child process is started in the background, and the function return immediately.

Usage

```
rcmd_bg(
  cmd,
  cmdargs = character(),
  libpath = .libPaths(),
  stdout = "|",
  stderr = "|",
  poll_connection = TRUE,
  repos = default_repos(),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  wd = ".",
  supervise = FALSE,
  ...
)
```

Arguments

`cmd`	Command to run. See `R --help` from the command line for the various commands. In the current version of R (3.2.4) these are: `BATCH`, `COMPILE`, `SHLIB`, `INSTALL`, `REMOVE`, `build`, `check`, `LINK`, `Rprof`, `Rdconv`, `Rd2pdf`, `Rd2txt`, `Stangle`, `Sweave`, `Rdiff`, `config`, `javareconf`, `rtags`.

`cmdargs`	Command line arguments.
`libpath`	The library path.
`stdout`	Optionally a file name to send the standard output to.
`stderr`	Optionally a file name to send the standard error to. It may be the same as `stdout`, in which case standard error is redirected to standard output. It can also be the special string `"2>&1"`, in which case standard error will be redirected to standard output.
`poll_connection`	Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the parent.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`wd`	Working directory to use for running the command. Defaults to the current working directory.
`supervise`	Whether to register the process with a supervisor. If `TRUE`, the supervisor will ensure that the process is killed when the R process exits.
`...`	Extra arguments are passed to the processx::process constructor.

Value

It returns a process object.

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

See Also

Other R CMD commands: `rcmd_copycat()`, `rcmd()`

rcmd_copycat — *Call and R CMD command, while mimicking the current R session*

Description

This function is similar to rcmd(), but it has slightly different defaults:

- The repos options is unchanged.
- No extra environment variables are defined.

Usage

```
rcmd_copycat(
  cmd,
  cmdargs = character(),
  libpath = .libPaths(),
  repos = getOption("repos"),
  env = character(),
  ...
)
```

Arguments

cmd	Command to run. See R --help from the command line for the various commands. In the current version of R (3.2.4) these are: BATCH, COMPILE, SHLIB, INSTALL, REMOVE, build, check, LINK, Rprof, Rdconv, Rd2pdf, Rd2txt, Stangle, Sweave, Rdiff, config, javareconf, rtags.
cmdargs	Command line arguments.
libpath	The library path.
repos	The repos option. If NULL, then no repos option is set. This options is only used if user_profile or system_profile is set FALSE, as it is set using the system or the user profile.
env	Environment variables to set for the child process.
...	Additional arguments are passed to rcmd().

Security considerations

callr makes a copy of the user's .Renviron file and potentially of the local or user .Rprofile, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

See Also

Other R CMD commands: rcmd_bg(), rcmd()

`rcmd_process` *External* `R CMD` *Process*

Description

An `R CMD *` command that runs in the background. This is an R6 class that extends the processx::process class.

Super class

`processx::process` -> `rcmd_process`

Methods

Public methods:

- `rcmd_process$new()`
- `rcmd_process$finalize()`
- `rcmd_process$clone()`

Method `new()`**:** Start an `R CMD` process.

Usage:

```
rcmd_process$new(options)
```

Arguments:

`options` A list of options created via `rcmd_process_options()`.

Returns: A new `rcmd_process` object.

Method `finalize()`**:** Clean up the temporary files created for an `R CMD` process.

Usage:

```
rcmd_process$finalize()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
rcmd_process$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
options <- rcmd_process_options(cmd = "config", cmdargs = "CC")
rp <- rcmd_process$new(options)
rp$wait()
rp$read_output_lines()
```

`rcmd_process_options` *Create options for an rcmd_process object*

Description

Create options for an rcmd_process object

Usage

```
rcmd_process_options(...)
```

Arguments

`...`	Options to override, named arguments.

Value

A list of options.

`rcmd_process_options()` creates a set of options to initialize a new object from the `rcmd_process` class. Its arguments must be named, the names are used as option names. The options correspond to (some of) the arguments of the `rcmd()` function. At least the `cmd` option must be specified, to select the `R CMD` subcommand to run. Typically `cmdargs` is specified as well, to supply more arguments to `R CMD`.

Examples

```
## List all options and their default values:
rcmd_process_options()
```

`rcmd_safe_env` *`rcmd_safe_env` returns a set of environment variables that are more appropriate for `rcmd_safe()`. It is exported to allow manipulating these variables (e.g. add an extra one), before passing them to the `rcmd()` functions.*

Description

It currently has the following variables:

- `CYGWIN="nodosfilewarning"`: On Windows, do not warn about MS-DOS style file names.
- `R_TESTS=""` This variable is set by `R CMD check`, and makes the child R process load a startup file at startup, from the current working directory, that is assumed to be the `/test` directory of the package being checked. If the current working directory is changed to something else (as it typically is by `testthat`, then R cannot start. Setting it to the empty string ensures that `callr` can be used from unit tests.
- `R_BROWSER="false"`: typically we don't want to start up a browser from the child R process.
- `R_PDFVIEWER="false"`: similarly for the PDF viewer.

Usage

```
rcmd_safe_env()
```

Details

Note that `callr` also sets the `R_ENVIRON`, `R_ENVIRON_USER`, `R_PROFILE` and `R_PROFILE_USER` environment variables appropriately, unless these are set by the user in the `env` argument of the `r`, etc. calls.

Value

A named character vector of environment variables.

`rscript` *Run an R script*

Description

It uses the `Rscript` program corresponding to the current R version, to run the script. It streams `stdout` and `stderr` of the process.

Usage

```
rscript(
  script,
  cmdargs = character(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = NULL,
  stderr = NULL,
  poll_connection = TRUE,
  echo = FALSE,
  show = TRUE,
  callback = NULL,
  block_callback = NULL,
  spinner = FALSE,
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  timeout = Inf,
  wd = ".",
  fail_on_status = TRUE,
  color = TRUE,
  ...
)
```

Arguments

`script`	Path of the script to run.
`cmdargs`	Command line arguments.
`libpath`	The library path.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`stdout`	Optionally a file name to send the standard output to.
`stderr`	Optionally a file name to send the standard error to. It may be the same as `stdout`, in which case standard error is redirected to standard output. It can also be the special string `"2>&1"`, in which case standard error will be redirected to standard output.
`poll_connection`	Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the parent.
`echo`	Whether to echo the complete command run by `rcmd`.
`show`	Logical, whether to show the standard output on the screen while the child process is running. Note that this is independent of the `stdout` and `stderr` arguments. The standard error is not shown currently.
`callback`	A function to call for each line of the standard output and standard error from the child process. It works together with the `show` option; i.e. if `show = TRUE`, and a callback is provided, then the output is shown of the screen, and the callback is also called.
`block_callback`	A function to call for each block of the standard output and standard error. This callback is not line oriented, i.e. multiple lines or half a line can be passed to the callback.
`spinner`	Whether to show a calming spinner on the screen while the child R session is running. By default it is shown if `show = TRUE` and the R session is interactive.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`timeout`	Timeout for the function call to finish. It can be a base::difftime object, or a real number, meaning seconds. If the process does not finish before the timeout period expires, then a `system_command_timeout_error` error is thrown. `Inf` means no timeout.
`wd`	Working directory to use for running the command. Defaults to the current working directory.
`fail_on_status`	Whether to throw an R error if the command returns with a non-zero status code. By default no error is thrown.
`color`	Whether to use terminal colors in the child process, assuming they are active in the parent process.
`...`	Extra arguments are passed to `processx::run()`.

Security considerations

callr makes a copy of the user's .Renviron file and potentially of the local or user .Rprofile, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

---

rscript_process *External* Rscript *process*

Description

An Rscript script.R command that runs in the background. This is an R6 class that extends the processx::process class.

Super class

processx::process -> rscript_process

Methods

Public methods:

- rscript_process$new()
- rscript_process$finalize()
- rscript_process$clone()

Method new()**:** Create a new Rscript process.

Usage:

```
rscript_process$new(options)
```

Arguments:

options A list of options created via rscript_process_options().

Method finalize()**:** Clean up after an Rsctipt process, remove temporary files.

Usage:

```
rscript_process$finalize()
```

Method clone()**:** The objects of this class are cloneable with this method.

Usage:

```
rscript_process$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
options <- rscript_process_options(script = "script.R")
rp <- rscript_process$new(options)
rp$wait()
rp$read_output_lines()
```

---

rscript_process_options

Create options for an rscript_process object

Description

Create options for an rscript_process object

Usage

```
rscript_process_options(...)
```

Arguments

`...`	Options to override, named arguments.

Value

A list of options.

`rscript_process_options()` creates a set of options to initialize a new object from the `rscript_process` class. Its arguments must be named, the names are used as option names. The options correspond to (some of) the arguments of the `rscript()` function. At least the `script` option must be specified, the script file to run.

Examples

```
## List all options and their default values:
rscript_process_options()
```

r_bg *Evaluate an expression in another R session, in the background*

Description

Starts evaluating an R function call in a background R process, and returns immediately. Use `p$get_result()` to collect the result or to throw an error if the background computation failed.

Usage

```
r_bg(
  func,
  args = list(),
  libpath = .libPaths(),
  repos = default_repos(),
  stdout = "|",
  stderr = "|",
  poll_connection = TRUE,
  error = getOption("callr.error", "error"),
  cmdargs = c("--slave", "--no-save", "--no-restore"),
  system_profile = FALSE,
  user_profile = "project",
  env = rcmd_safe_env(),
  supervise = FALSE,
  package = FALSE,
  arch = "same",
  ...
)
```

Arguments

`func` Function object to call in the new R process. The function should be self-contained and only refer to other functions and use variables explicitly from other packages using the `::` notation. By default the environment of the function is set to `.GlobalEnv` before passing it to the child process. (See the `package` option if you want to keep the environment.) Because of this, it is good practice to create an anonymous function and pass that to `callr`, instead of passing a function object from a (base or other) package. In particular

```
r(.libPaths)
```

does not work, because `.libPaths` is defined in a special environment, but

```
r(function() .libPaths())
```

works just fine.

`args` Arguments to pass to the function. Must be a list.

`libpath` The library path.

`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`stdout`	The name of the file the standard output of the child R process will be written to. If the child process runs with the `--slave` option (the default), then the commands are not echoed and will not be shown in the standard output. Also note that you need to call `print()` explicitly to show the output of the command(s). IF `NULL`, then standard output is not returned, but it is recorded and included in the error object if an error happens. Various special values for this argument such as `"|"` are explained in the `stdout` argument of processx::process.
`stderr`	The name of the file the standard error of the child R process will be written to. In particular `message()` sends output to the standard error. If nothing was sent to the standard error, then this file will be empty. This argument can be the same file as `stdout`, in which case they will be correctly interleaved. If this is the string `"2>&1"`, then standard error is redirected to standard output. IF `NULL`, then standard output is not returned, but it is recorded and included in the error object if an error happens. Various special values for this argument such as `"|"` are explained in the `stdout` argument of processx::process.
`poll_connection`	Whether to have a control connection to the process. This is used to transmit messages from the subprocess to the main process.
`error`	What to do if the remote process throws an error. See details below.
`cmdargs`	Command line arguments to pass to the R process. Note that `c("-f", rscript)` is appended to this, `rscript` is the name of the script file to run. This contains a call to the supplied function and some error handling code.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`supervise`	Whether to register the process with a supervisor. If `TRUE`, the supervisor will ensure that the process is killed when the R process exits.
`package`	Whether to keep the environment of `func` when passing it to the other package. Possible values are: • `FALSE`: reset the environment to `.GlobalEnv`. This is the default. • `TRUE`: keep the environment as is. • pkg: set the environment to the pkg package namespace.
`arch`	Architecture to use in the child process, for multi-arch builds of R. By default the same as the main process. See `supported_archs()`. If it contains a forward or backward slash character, then it is taken as the path to the R executable. Note that on Windows you need the path to `Rterm.exe`.
`...`	Extra arguments are passed to the processx::process constructor.

Value

An `r_process` object, which inherits from process, so all `process` methods can be called on it, and in addition it also has a `get_result()` method to collect the result.

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

Examples

```
rx <- r_bg(function() 1 + 2)

# wait until it is done
rx$wait()
rx$is_alive()
rx$get_result()
```

`r_copycat` *Run an R process that mimics the current R process*

Description

Differences to `r()`:

- No extra repositories are set up.
- The `--no-save`, `--no-restore` command line arguments are not used. (But `--slave` still is.)
- The system profile and the user profile are loaded.
- No extra environment variables are set up.

Usage

```
r_copycat(
  func,
  args = list(),
  libpath = .libPaths(),
  repos = getOption("repos"),
  cmdargs = "--slave",
  system_profile = TRUE,
  user_profile = TRUE,
  env = character(),
  ...
)
```

Arguments

`func`	Function object to call in the new R process. The function should be self-contained and only refer to other functions and use variables explicitly from other packages using the `::` notation. By default the environment of the function is set to `.GlobalEnv` before passing it to the child process. (See the `package` option if you want to keep the environment.) Because of this, it is good practice to create an anonymous function and pass that to `callr`, instead of passing a function object from a (base or other) package. In particular `r(.libPaths)` does not work, because `.libPaths` is defined in a special environment, but `r(function() .libPaths())` works just fine.
`args`	Arguments to pass to the function. Must be a list.
`libpath`	The library path.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.
`cmdargs`	Command line arguments to pass to the R process. Note that `c("-f", rscript)` is appended to this, `rscript` is the name of the script file to run. This contains a call to the supplied function and some error handling code.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`...`	Additional arguments are passed to `r()`.

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

See Also

Other callr functions: `r_vanilla()`, `r()`

`r_process` *External R Process*

Description

An R process that runs in the background. This is an R6 class that extends the processx::process class. The process starts in the background, evaluates an R function call, and then quits.

Super class

`processx::process` -> `r_process`

Methods

Public methods:

- `r_process$new()`
- `r_process$get_result()`
- `r_process$finalize()`
- `r_process$clone()`

Method `new()`**:** Start a new R process in the background.

Usage:

```
r_process$new(options)
```

Arguments:

`options` A list of options created via `r_process_options()`.

Returns: A new `r_process` object.

Method `get_result()`**:** Return the result, an R object, from a finished background R process. If the process has not finished yet, it throws an error. (You can use `wait()` method (see processx::process) to wait for the process to finish, optionally with a timeout.) You can also use `processx::poll()` to wait for the end of the process, together with other processes or events.

Usage:

```
r_process$get_result()
```

Returns: The return value of the R expression evaluated in the R process.

Method `finalize()`**:** Clean up temporary files once an R process has finished and its handle is garbage collected.

Usage:

```
r_process$finalize()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
r_process$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## List all options and their default values:
r_process_options()

## Start an R process in the background, wait for it, get result
opts <- r_process_options(func = function() 1 + 1)
rp <- r_process$new(opts)
rp$wait()
rp$get_result()
```

`r_process_options` *Create options for an r_process object*

Description

Create options for an r_process object

Usage

```
r_process_options(...)
```

Arguments

`...`	Options to override, named arguments.

Value

A list of options.

`r_process_options()` creates a set of options to initialize a new object from the `r_process` class. Its arguments must be named, the names are used as option names. The options correspond to (some of) the arguments of the `r()` function. At least the `func` option must be specified, this is the R function to run in the background.

Examples

```
## List all options and their default values:
r_process_options()
```

`r_session` *External R Session*

Description

A permanent R session that runs in the background. This is an R6 class that extends the processx::process class.

The process is started at the creation of the object, and then it can be used to evaluate R function calls, one at a time.

Super class

`processx::process` -> `r_session`

Public fields

`status` Status codes returned by `read()`.

Methods

Public methods:

- `r_session$new()`
- `r_session$run()`
- `r_session$run_with_output()`
- `r_session$call()`
- `r_session$poll_process()`
- `r_session$get_state()`
- `r_session$get_running_time()`
- `r_session$read()`
- `r_session$close()`
- `r_session$traceback()`
- `r_session$debug()`
- `r_session$attach()`
- `r_session$finalize()`
- `r_session$print()`
- `r_session$clone()`

Method `new()`**:** creates a new R background process. It can wait for the process to start up (`wait = TRUE`), or return immediately, i.e. before the process is actually ready to run. In the latter case you may call the `poll_process()` method to make sure it is ready.

Usage:

```
r_session$new(options = r_session_options(), wait = TRUE, wait_timeout = 3000)
```

Arguments:

`options` A list of options created via `r_session_options()`.

`wait` Whether to wait for the R process to start and be ready for running commands.

`wait_timeout` Timeout for waiting for the R process to start, in milliseconds.

Returns: An `r_session` object.

Method `run()`**:** Similar to `r()`, but runs the function in a permanent background R session. It throws an error if the function call generated an error in the child process.

Usage:

```
r_session$run(func, args = list(), package = FALSE)
```

Arguments:

`func` Function object to call in the background R process. Please read the notes for the similar argument of `r()`.

`args` Arguments to pass to the function. Must be a list.

`package` Whether to keep the environment of `func` when passing it to the other package. Possible values are:

- `FALSE`: reset the environment to `.GlobalEnv`. This is the default.
- `TRUE`: keep the environment as is.
- `pkg`: set the environment to the `pkg` package namespace.

Returns: The return value of the R expression.

Method `run_with_output()`**:** Similar to `$run()`, but returns the standard output and error of the child process as well. It does not throw on errors, but returns a non-`NULL` `error` member in the result list.

Usage:

```
r_session$run_with_output(func, args = list(), package = FALSE)
```

Arguments:

`func` Function object to call in the background R process. Please read the notes for the similar argument of `r()`.

`args` Arguments to pass to the function. Must be a list.

`package` Whether to keep the environment of `func` when passing it to the other package. Possible values are:

- `FALSE`: reset the environment to `.GlobalEnv`. This is the default.
- `TRUE`: keep the environment as is.
- `pkg`: set the environment to the `pkg` package namespace.

Returns: A list with the following entries.

- `result`: The value returned by `func`. On error this is `NULL`.
- `stdout`: The standard output of the process while evaluating
- `stderr`: The standard error of the process while evaluating the `func` call.
- `error`: On error it contains an error object, that contains the error thrown in the subprocess. Otherwise it is `NULL`.
- `code`, `message`: These fields are used by call internally and you can ignore them.

Method `call()`**:** Starts running a function in the background R session, and returns immediately. To check if the function is done, call the `poll_process()` method.

Usage:

```
r_session$call(func, args = list(), package = FALSE)
```

Arguments:

`func` Function object to call in the background R process. Please read the notes for the similar argument of `r()`.

`args` Arguments to pass to the function. Must be a list.

`package` Whether to keep the environment of `func` when passing it to the other package. Possible values are:

- `FALSE`: reset the environment to `.GlobalEnv`. This is the default.
- `TRUE`: keep the environment as is.
- `pkg`: set the environment to the `pkg` package namespace.

Method `poll_process()`**:** Poll the R session with a timeout. If the session has finished the computation, it returns with `"ready"`. If the timeout is reached, it returns with `"timeout"`.

Usage:

```
r_session$poll_process(timeout)
```

Arguments:

`timeout` Timeout period in milliseconds.

Returns: Character string `"ready"` or `"timeout"`.

Method `get_state()`**:** Return the state of the R session.

Usage:

```
r_session$get_state()
```

Returns: Possible values:

- `"starting"`: starting up,
- `"idle"`: ready to compute,
- `"busy"`: computing right now,
- `"finished"`: the R process has finished.

Method `get_running_time()`**:** Returns the elapsed time since the R process has started, and the elapsed time since the current computation has started. The latter is `NA` if there is no active computation.

Usage:

```
r_session$get_running_time()
```

Returns: Named vector of `POSIXct` objects. The names are `"total"` and `"current"`.

Method `read()`**:** Reads an event from the child process, if there is one available. Events might signal that the function call has finished, or they can be progress report events.

This is a low level function that you only need to use if you want to process events (messages) from the R session manually.

Usage:

```
r_session$read()
```

Returns: `NULL` if no events are available. Otherwise a named list, which is also a `callr_session_result` object. The list always has a `code` entry which is the type of the event. See also `r_session$public_fields$status` for symbolic names of the event types.

- `200`: (`DONE`) The computation is done, and the event includes the result, in the same form as for the `run()` method.
- `201`: (`STARTED`) An R session that was in 'starting' state is ready to go.
- `202`: (`ATTACH_DONE`) Used by the `attach()` method.
- `301`: (`MSG`) A message from the subprocess. The message is a condition object with class `callr_message`. (It typically has other classes, e.g. `cli_message` for output from the cli package.)
- `500`: (`EXITED`) The R session finished cleanly. This means that the evaluated expression quit R.
- `501`: (`CRASHED`) The R session crashed or was killed.
- `502`: (`CLOSED`) The R session closed its end of the connection that callr uses for communication.

Method `close()`**:** Terminate the current computation and the R process. The session object will be in `"finished"` state after this.

Usage:

```
r_session$close(grace = 1000)
```

Arguments:

`grace` Grace period in milliseconds, to wait for the subprocess to exit cleanly, after its standard input is closed. If the process is still running after this period, it will be killed.

Method `traceback()`**:** The `traceback()` method can be used after an error in the R subprocess. It is equivalent to the `base::traceback()` call, in the subprocess.

On callr version 3.8.0 and above, you need to set the `callr.traceback` option to `TRUE` (in the main process) to make the subprocess save the trace on error. This is because saving the trace can be costly for large objects passed as arguments.

Usage:

```
r_session$traceback()
```

Returns: The same output as from `base::traceback()`

Method `debug()`**:** Interactive debugger to inspect the dumped frames in the subprocess, after an error. See more at r_session_debug.

On callr version 3.8.0 and above, you need to set the `callr.traceback` option to `TRUE` (in the main process) to make the subprocess dump frames on error. This is because saving the frames can be costly for large objects passed as arguments.

Usage:

```
r_session$debug()
```

Method `attach()`**:** Experimental function that provides a REPL (Read-Eval-Print-Loop) to the subprocess.

Usage:

```
r_session$attach()
```

Method `finalize()`**:** Finalizer that is called when garbage collecting an `r_session` object, to clean up temporary files.

Usage:

```
r_session$finalize()
```

Method `print()`**:** Print method for an `r_session`.

Usage:

```
r_session$print(...)
```

Arguments:

`...` Arguments are not used currently.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
r_session$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
rs <- r_ression$new()

rs$run(function() 1 + 2)

rs$call(function() Sys.sleep(1))
rs$get_state()

rs$poll_process(-1)
rs$get_state()
rs$read()
```

`r_session_debug` *Interactive debugging of persistent R sessions*

Description

The `r_session$debug()` method is an interactive debugger to inspect the stack of the background process after an error.

Details

Note that on callr version 3.8.0 and above, you need to set the `callr.traceback` option to `TRUE` (in the main process) to make the subprocess dump the frames on error. This is because saving the frames can be costly for large objects passed as arguments.

`$debug()` starts a REPL (Read-Eval-Print-Loop), that evaluates R expressions in the subprocess. It is similar to `browser()` and `debugger()` and also has some extra commands:

- `.help` prints a short help message.
- `.where` prints the complete stack trace of the error. (The same as the `$traceback()` method.
- `.inspect <n>` switches the "focus" to frame <n>. Frame 0 is the global environment, so `.inspect 0` will switch back to that.

To exit the debugger, press the usual interrupt key, i.e. `CTRL+c` or `ESC` in some GUIs.

Here is an example session that uses `$debug()` (some output is omitted for brevity):

```
# ----------------------------------------------------------------------
> rs <- r_session$new()
> rs$run(function() knitr::knit("no-such-file"))
Error in rs_run(self, private, func, args) :
 callr subprocess failed: cannot open the connection

> rs$debug()
Debugging in process 87361, press CTRL+C (ESC) to quit. Commands:
  .where       -- print stack trace
  .inspect <n> -- inspect a frame, 0 resets to .GlobalEnv
  .help        -- print this message
  <cmd>        -- run <cmd> in frame or .GlobalEnv

3: file(con, "r")
2: readLines(input2, encoding = "UTF-8", warn = FALSE)
1: knitr::knit("no-such-file") at #1

RS 87361 > .inspect 1

RS 87361 (frame 1) > ls()
 [1] "encoding"  "envir"     "ext"       "in.file"   "input"     "input.dir"
 [7] "input2"    "ocode"     "oconc"     "oenvir"    "oopts"     "optc"
[13] "optk"      "otangle"   "out.purl"  "output"    "quiet"     "tangle"
[19] "text"

RS 87361 (frame 1) > input
[1] "no-such-file"

RS 87361 (frame 1) > file.exists(input)
[1] FALSE

RS 87361 (frame 1) > # <CTRL + C>
# ----------------------------------------------------------------------
```

`r_session_options` *Create options for an r_session object*

Description

Create options for an r_session object

Usage

```
r_session_options(...)
```

Arguments

`...`	Options to override, named arguments.

Value

Named list of options.

The current options are:

- `libpath`: Library path for the subprocess. By default the same as the *current* library path. I.e. *not* necessarily the library path of a fresh R session.)
- `repos`: `repos` option for the subprocess. By default the current value of the main process.
- `stdout`: Standard output of the sub-process. This can be `NULL` or a pipe: `"|"`. If it is a pipe then the output of the subprocess is not included in the responses, but you need to poll and read it manually. This is for experts. Note that this option is not used for the startup phase that currently always runs with `stdout = "|"`.
- `stderr`: Similar to `stdout`, but for the standard error. Like `stdout`, it is not used for the startup phase, which runs with `stderr = "|"`.
- `error`: See 'Error handling' in `r()`.
- `cmdargs`: See the same argument of `r()`. (Its default might be different, though.)
- `system_profile`: See the same argument of `r()`.
- `user_profile`: See the same argument of `r()`.
- `env`: See the same argument of `r()`.
- `load_hook`: `NULL`, or code (quoted) to run in the sub-process at start up. (I.e. not for every single `run()` call.)
- `extra`: List of extra arguments to pass to processx::process.

Call `r_session_options()` to see the default values. `r_session_options()` might contain undocumented entries, you cannot change these.

Examples

```
r_session_options()
```

r_vanilla *Run an R child process, with no configuration*

Description

It tries to mimic a fresh R installation. In particular:

- No library path setting.
- No CRAN(-like) repository is set.
- The system and user profiles are not run.

Usage

```
r_vanilla(
  func,
  args = list(),
  libpath = character(),
  repos = c(CRAN = "@CRAN@"),
  cmdargs = "--slave",
  system_profile = FALSE,
  user_profile = FALSE,
  env = character(),
  ...
)
```

Arguments

`func`	Function object to call in the new R process. The function should be self-contained and only refer to other functions and use variables explicitly from other packages using the `::` notation. By default the environment of the function is set to `.GlobalEnv` before passing it to the child process. (See the `package` option if you want to keep the environment.) Because of this, it is good practice to create an anonymous function and pass that to `callr`, instead of passing a function object from a (base or other) package. In particular
	
	`r(.libPaths)`
	
	does not work, because `.libPaths` is defined in a special environment, but
	
	`r(function() .libPaths())`
	
	works just fine.

`args`	Arguments to pass to the function. Must be a list.
`libpath`	The library path.
`repos`	The `repos` option. If `NULL`, then no `repos` option is set. This options is only used if `user_profile` or `system_profile` is set `FALSE`, as it is set using the system or the user profile.

`cmdargs`	Command line arguments to pass to the R process. Note that `c("-f", rscript)` is appended to this, `rscript` is the name of the script file to run. This contains a call to the supplied function and some error handling code.
`system_profile`	Whether to use the system profile file.
`user_profile`	Whether to use the user's profile file. If this is `"project"`, then only the profile from the working directory is used, but the `R_PROFILE_USER` environment variable and the user level profile are not. See also "Security considerations" below.
`env`	Environment variables to set for the child process.
`...`	Additional arguments are passed to `r()`.

Security considerations

`callr` makes a copy of the user's `.Renviron` file and potentially of the local or user `.Rprofile`, in the session temporary directory. Avoid storing sensitive information such as passwords, in your environment file or your profile, otherwise this information will get scattered in various files, at least temporarily, until the subprocess finishes. You can use the keyring package to avoid passwords in plain files.

See Also

Other callr functions: `r_copycat()`, `r()`

Examples

```
# Compare to r()
r(function() .libPaths())
r_vanilla(function() .libPaths())

r(function() getOption("repos"))
r_vanilla(function() getOption("repos"))
```

`supported_archs` *Find supported sub-architectures for the current R installation*

Description

This function uses a heuristic, which might fail, so its result should be taken as a best guess.

Usage

```
supported_archs()
```

Value

Character vector of supported architectures. If the current R build is not a multi-architecture build, then an empty string scalar is returned.

Examples

```
supported_archs()
```

Index