

# Package ‘sys’

March 10, 2019

**Type** Package

**Title** Powerful and Reliable Tools for Running System Commands in R

**Version** 3.1

**Description** Drop-in replacements for the base `system2()` function with fine control and consistent behavior across platforms. Supports clean interruption, timeout, background tasks, and streaming STDIN / STDOUT / STDERR over binary or text connections. Arguments on Windows automatically get encoded and quoted to work on different locales.

**License** MIT + file LICENSE

**URL** <https://github.com/jeroen/sys#readme>

**BugReports** <https://github.com/jeroen/sys/issues>

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**Suggests** unix (>= 1.4), spelling, testthat

**Language** en-US

**NeedsCompilation** yes

**Author** Jeroen Ooms [aut, cre] (<<https://orcid.org/0000-0002-4035-0289>>),  
Gábor Csárdi [ctb]

**Maintainer** Jeroen Ooms <[jeroen@berkeley.edu](mailto:jeroen@berkeley.edu)>

**Repository** CRAN

**Date/Publication** 2019-03-10 20:40:03 UTC

## R topics documented:

<code>as_text</code> . . . . .	2
<code>exec</code> . . . . .	2
<code>exec_r</code> . . . . .	5
<code>quote</code> . . . . .	6
<code>sys-deprecated</code> . . . . .	6

<b>Index</b>	<b>7</b>
--------------	----------

as\_text

*Convert Raw to Text*

---

**Description**

Parses a raw vector as lines of text. This is similar to [charToRaw](#) but splits output by (platform specific) linebreaks and allows for marking output with a given encoding.

**Usage**

```
as_text(x, ...)
```

**Arguments**

x	vector to be converted to text
...	parameters passed to <a href="#">readLines</a> such as encoding or n

**See Also**

[base::charToRaw](#)

---

exec

*Running System Commands*

---

**Description**

Powerful replacements for [system2](#) with support for interruptions, background tasks and fine grained control over STDOUT / STDERR binary or text streams.

**Usage**

```
exec_wait(cmd, args = NULL, std_out = stdout(), std_err = stderr(),  
          std_in = NULL, timeout = 0)
```

```
exec_background(cmd, args = NULL, std_out = TRUE, std_err = TRUE,  
               std_in = NULL)
```

```
exec_internal(cmd, args = NULL, std_in = NULL, error = TRUE,  
              timeout = 0)
```

```
exec_status(pid, wait = TRUE)
```

**Arguments**

<code>cmd</code>	the command to run. Either a full path or the name of a program on the PATH. On Windows this is automatically converted to a short path using <code>Sys.which</code> , unless wrapped in <code>I()</code> .
<code>args</code>	character vector of arguments to pass. On Windows these automatically get quoted using <code>windows_quote</code> , unless the value is wrapped in <code>I()</code> .
<code>std_out</code>	if and where to direct child process STDOUT. Must be one of TRUE, FALSE, file-name, connection object or callback function. See section on <i>Output Streams</i> below for details.
<code>std_err</code>	if and where to direct child process STDERR. Must be one of TRUE, FALSE, file-name, connection object or callback function. See section on <i>Output Streams</i> below for details.
<code>std_in</code>	file path to map <code>std_in</code>
<code>timeout</code>	maximum time in seconds
<code>error</code>	automatically raise an error if the exit status is non-zero.
<code>pid</code>	integer with a process ID
<code>wait</code>	block until the process completes

**Details**

Each value within the `args` vector will automatically be quoted when needed; you should not quote arguments yourself. Doing so anyway could lead to the value being quoted twice on some platforms.

The `exec_wait` function runs a system command and waits for the child process to exit. When the child process completes normally (either success or error) it returns with the program exit code. Otherwise (if the child process gets aborted) R raises an error. The R user can interrupt the program by sending SIGINT (press ESC or CTRL+C) in which case the child process tree is properly terminated. Output streams STDOUT and STDERR are piped back to the parent process and can be sent to a connection or callback function. See the section on *Output Streams* below for details.

The `exec_background` function starts the program and immediately returns the PID of the child process. This is useful for running a server daemon or background process. Because this is non-blocking, `std_out` and `std_err` can only be TRUE/FALSE or a file path. The state of the process can be checked with `exec_status` which returns the exit status, or NA if the process is still running. If `wait = TRUE` then `exec_status` blocks until the process completes (but can be interrupted). The child can be killed with `tools::pskill`.

The `exec_internal` function is a convenience wrapper around `exec_wait` which automatically captures output streams and raises an error if execution fails. Upon success it returns a list with status code, and raw vectors containing stdout and stderr data (use `as_text` for converting to text).

**Value**

`exec_background` returns a `pid`. `exec_wait` returns an exit code. `exec_internal` returns a list with exit code, stdout and stderr strings.

## Output Streams

The `std_out` and `std_err` parameters are used to control how output streams of the child are processed. Possible values for both foreground and background processes are:

- `TRUE`: print child output in R console
- `FALSE`: suppress output stream
- *string*: name or path of file to redirect output

In addition the `exec_wait` function also supports the following `std_out` and `std_err` types:

- *connection* a writable R [connection](#) object such as `stdout` or `stderr`
- *function*: callback function with one argument accepting a raw vector (use [as\\_text](#) to convert to text).

When using `exec_background` with `std_out = TRUE` or `std_err = TRUE` on Windows, separate threads are used to print output. This works in RStudio and RTerm but not in RGui because the latter has a custom I/O mechanism. Directing output to a file is usually the safest option.

## See Also

Base [system2](#) and [pipe](#) provide other methods for running a system command with output.

Other sys: [exec\\_r](#)

## Examples

```
# Run a command (interrupt with CTRL+C)
status <- exec_wait("date")

# Capture std/out
out <- exec_internal("date")
print(out$status)
cat(as_text(out$stdout))

if(nchar(Sys.which("ping"))){

# Run a background process (daemon)
pid <- exec_background("ping", "localhost")

# Kill it after a while
Sys.sleep(2)
tools::pskill(pid)

# Cleans up the zombie proc
exec_status(pid)
rm(pid)
}
```

---

exec_r	<i>Execute R from R</i>
--------	-------------------------

---

### Description

Convenience wrappers for [exec\\_wait](#) and [exec\\_internal](#) that shell out to R itself: `R.home("bin/R")`.

### Usage

```
r_wait(args = "--vanilla", stdout = stdout(), stderr = stderr(),
        stdin = NULL)
```

```
r_internal(args = "--vanilla", stdin = NULL, error = TRUE)
```

```
r_background(args = "--vanilla", stdout = TRUE, stderr = TRUE,
              stdin = NULL)
```

### Arguments

args	command line arguments for R
stdout	if and where to direct child process STDOUT. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on <i>Output Streams</i> below for details.
stderr	if and where to direct child process STDERR. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on <i>Output Streams</i> below for details.
stdin	a file to send to stdin, usually an R script (see examples).
error	automatically raise an error if the exit status is non-zero.

### Details

This is a simple but robust way to invoke R commands in a separate process. Use the [callr](#) package if you need more sophisticated control over (multiple) R process jobs.

### See Also

Other sys: [exec](#)

### Examples

```
# Hello world
r_wait("--version")

# Run some code
r_wait(c('--vanilla', '-q', '-e', 'sessionInfo()'))

# Run a script via stdin
```

```
tmp <- tempfile()
writeLines(c("x <- rnorm(100)", "mean(x)"), con = tmp)
r_wait(std_in = tmp)
```

---

quote	<i>Quote arguments on Windows</i>
-------	-----------------------------------

---

### Description

Quotes and escapes shell arguments when needed so that they get properly parsed by most Windows programs. This function is used internally to automatically quote system commands, the user should normally not quote arguments manually.

### Usage

```
windows_quote(args)
```

### Arguments

args            character vector with arguments

### Details

Algorithm is ported to R from [libuv](#).

---

sys-deprecated	<i>Deprecated functions</i>
----------------	-----------------------------

---

### Description

These functions have moved into the `unix` package. Please update your references.

### Usage

```
eval_safe(...)
```

```
eval_fork(...)
```

### Arguments

...            see respective functions in the `unix` package

# Index

`as_text`, [2](#), [3](#), [4](#)

`base::charToRaw`, [2](#)

`charToRaw`, [2](#)

`connection`, [4](#)

`eval_fork` (sys-deprecated), [6](#)

`eval_safe` (sys-deprecated), [6](#)

`exec`, [2](#), [5](#)

`exec_background` (`exec`), [2](#)

`exec_internal`, [5](#)

`exec_internal` (`exec`), [2](#)

`exec_r`, [4](#), [5](#)

`exec_status` (`exec`), [2](#)

`exec_wait`, [5](#)

`exec_wait` (`exec`), [2](#)

`I()`, [3](#)

`pipe`, [4](#)

`quote`, [6](#)

`r_background` (`exec_r`), [5](#)

`r_internal` (`exec_r`), [5](#)

`r_wait` (`exec_r`), [5](#)

`readLines`, [2](#)

`stderr`, [4](#)

`stdout`, [4](#)

`sys` (`exec`), [2](#)

sys-deprecated, [6](#)

`Sys.which`, [3](#)

`system2`, [2](#), [4](#)

`tools::pskill`, [3](#)

`windows_quote`, [3](#)

`windows_quote` (`quote`), [6](#)