

# Package ‘JuliaCall’

October 12, 2022

**Type** Package

**Title** Seamless Integration Between R and 'Julia'

**Version** 0.17.5

**Date** 2022-09-08

**Description** Provides an R interface to 'Julia', which is a high-level, high-performance dynamic programming language for numerical computing, see [<https://julialang.org/>](https://julialang.org/) for more information. It provides a high-level interface as well as a low-level interface. Using the high level interface, you could call any 'Julia' function just like any R function with automatic type conversion. Using the low level interface, you could deal with C-level SEXP directly while enjoying the convenience of using a high-level programming language like 'Julia'.

**Depends** R (>= 3.4.0)

**License** MIT + file LICENSE

**URL** <https://github.com/Non-Contradiction/JuliaCall>

**BugReports** <https://github.com/Non-Contradiction/JuliaCall/issues>

**Encoding** UTF-8

**Imports** utils, Rcpp (>= 0.12.7), knitr (>= 1.28), rjson

**RoxygenNote** 7.1.2

**LinkingTo** Rcpp

**NeedsCompilation** yes

**ByteCompile** yes

**SystemRequirements** Julia >= 0.6.0, RCall.jl

**Suggests** testthat, rmarkdown, rappdirs, sass

**VignetteBuilder** knitr

**Author** Changcheng Li [aut, cre],  
Randy Lai [ctb],  
Dmitri Grominski [ctb],  
Nagi Teramo [ctb]

**Maintainer** Changcheng Li <cx1508@psu.edu>

Repository CRAN

Date/Publication 2022-09-08 11:23:04 UTC

## R topics documented:

autowrap . . . . .	2
call . . . . .	3
eng_juliacall . . . . .	4
install_julia . . . . .	4
JuliaCall . . . . .	5
JuliaObject . . . . .	6
JuliaObjectFields . . . . .	7
julia_assign . . . . .	7
julia_command . . . . .	8
julia_console . . . . .	9
julia_eval . . . . .	9
julia_exists . . . . .	10
julia_help . . . . .	10
julia_markdown_setup . . . . .	11
julia_notebook_setup . . . . .	11
julia_package . . . . .	12
julia_pkg_wrap . . . . .	13
julia_setup . . . . .	13
julia_source . . . . .	15
plotsViewer . . . . .	15
%>J% . . . . .	15
<b>Index</b>	<b>17</b>

---

autowrap	<i>Use automatic wrapper for julia type.</i>
----------	--

---

### Description

autowrap tells ‘JuliaCall’ to use automatic wrapper for julia type.

### Usage

```
autowrap(type, fields = NULL, methods = c())
```

### Arguments

type	the julia type to wrap.
fields	names of fields to be included in the wrapper. If the value is NULL, then every julia fields will be included in the wrapper.
methods	names of methods to be overloaded for the wrapper.

---

call	<i>Call julia functions.</i>
------	------------------------------

---

**Description**

`julia_do.call` is the `do.call` for julia. And `julia_call` calls julia functions. For usage of these functions, see documentation of arguments and examples.

**Usage**

```
julia_do.call(
  func_name,
  arg_list,
  need_return = c("R", "Julia", "None"),
  show_value = FALSE
)
```

```
julia_call(
  func_name,
  ...,
  need_return = c("R", "Julia", "None"),
  show_value = FALSE
)
```

**Arguments**

<code>func_name</code>	the name of julia function you want to call. If you add "." after 'func_name', the julia function call will be broadcasted.
<code>arg_list</code>	the list of the arguments you want to pass to the julia function.
<code>need_return</code>	whether you want julia to return value as an R object, a wrapper for julia object or no return. The value of <code>need_return</code> could be <code>TRUE</code> (equal to option "R") or <code>FALSE</code> (equal to option "None"), or one of the options "R", "Julia" and "None".
<code>show_value</code>	whether to invoke the julia display system or not.
<code>...</code>	the arguments you want to pass to the julia function.

**Details**

Note that named arguments will be discarded if the call uses dot notation, for example, `"sqrt."`.

**Examples**

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_do.call("sqrt", list(2))
  julia_call("sqrt", 2)
}
```

```
julia_call("sqrt.", 1:10)
}
```

---

eng\_juliacall

*Julia language engine in R Markdown*


---

### Description

Julia language engine in R Markdown

### Usage

```
eng_juliacall(options)
```

### Arguments

options            a list of chunk options

### Examples

```
knitr::knit_engines$set(julia = JuliaCall::eng_juliacall)
```

---

install\_julia

*Install Julia.*


---

### Description

Install Julia.

### Usage

```
install_julia(version = "latest", prefix = julia_default_install_dir())
```

### Arguments

version            The version of Julia to install (e.g. "1.6.3"). Defaults to "latest", which will install the most recent stable release.

prefix            the directory where Julia will be installed. If not set, a default location will be determined by rappdirs if it is installed, otherwise an error will be raised.

## Description

JuliaCall provides you with functions to call Julia functions and to use Julia packages as easy as possible.

## Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## The examples are quite time consuming

  ## Do initiation for JuliaCall and automatic installation if necessary

  julia <- julia_setup(installJulia = TRUE)

  ## Different ways for calculating `sqrt(2)`

  # julia$command("a = sqrt(2)"); julia$eval("a")
  julia_command("a = sqrt(2)"); julia_eval("a")

  # julia$eval("sqrt(2)")
  julia_eval("sqrt(2)")

  # julia$call("sqrt", 2)
  julia_call("sqrt", 2)

  # julia$eval("sqrt")(2)
  julia_eval("sqrt")(2)

  ## You can use `julia_exists` as `exists` in R to test
  ## whether a function or name exists in Julia or not

  # julia$exists("sqrt")
  julia_exists("sqrt")

  ## You can use `julia$help` to get help for Julia functions

  # julia$help("sqrt")
  julia_help("sqrt")

  ## You can install and use Julia packages through JuliaCall

  # julia$install_package("Optim")
  julia_install_package("Optim")

  # julia$install_package_if_needed("Optim")
  julia_install_package_if_needed("Optim")
```

```
# julia$installed_package("Optim")
julia_installed_package("Optim")

# julia$library("Optim")
julia_library("Optim")
}
```

---

JuliaObject

*Convert an R Object to Julia Object.*

---

### Description

JuliaObject converts an R object to julia object and returns a reference of the corresponding julia object.

### Usage

```
JuliaObject(x)
```

### Arguments

x                    the R object you want to convert to julia object.

### Value

an environment of class JuliaObject, which contains an id corresponding to the actual julia object.

### Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  a <- JuliaObject(1)
}
```

---

JuliaObjectFields      *JuliaObject Fields.*

---

### Description

Get the field names, get or set certain fields of an JuliaObject.

### Usage

```
fields(object)

## S3 method for class 'JuliaObject'
fields(object)

field(object, name)

## S3 method for class 'JuliaObject'
field(object, name)

field(object, name) <- value

## S3 replacement method for class 'JuliaObject'
field(object, name) <- value
```

### Arguments

object	the JuliaObject.
name	a character string specifying the fields to be accessed or set.
value	the new value of the field of the JuliaObject.

---

julia\_assign      *Assign a value to a name in julia.*

---

### Description

julia\_assign assigns a value to a name in julia with automatic type conversion.

### Usage

```
julia_assign(x, value)
```

### Arguments

x	a variable name, given as a character string.
value	a value to be assigned to x, note that R value will be converted to corresponding julia value automatically.

## Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_assign("x", 2)
  julia_assign("rsqrt", sqrt)
}
```

---

julia_command	<i>Evaluate string commands in julia and (may) invoke the julia display system.</i>
---------------	---

---

## Description

julia\_command evaluates string commands in julia without returning the result back to R. However, it may evoke julia display system, see the documentation of the argument 'show\_value' for more details. If you need to get the evaluation result in R, you can use julia\_eval.

## Usage

```
julia_command(cmd, show_value = !endsWith(trimws(cmd, "right"), ";"))
```

## Arguments

cmd	the command string you want to evaluate in julia.
show_value	whether to display julia returning value or not, the default value is 'FALSE' if the 'cmd' ends with semicolon and 'TRUE' otherwise.

## Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_command("a = sqrt(2);")
}
```



---

julia_console	<i>Open julia console.</i>
---------------	----------------------------

---

**Description**

Open julia console.

**Usage**

```
julia_console()
```

**Examples**

```
if (identical(interactive(), TRUE)) { ## julia_setup is quite time consuming
  julia_console()
}
```

---

julia_eval	<i>Evaluate string commands in julia and get the result back in R.</i>
------------	--

---

**Description**

`julia_eval` evaluates string commands in julia and returns the result to R. The returning julia object will be automatically converted to an R object or a `JuliaObject` wrapper, see the documentation of the argument `'need_return'` for more details. `'julia_eval'` will not invoke julia display system. If you don't need the returning result in R or you want to invoke the julia display system, you can use `julia_command`.

**Usage**

```
julia_eval(cmd, need_return = c("R", "Julia"))
```

**Arguments**

<code>cmd</code>	the command string you want to evaluate in julia.
<code>need_return</code>	whether you want julia to return value as an R object or a wrapper for julia object.

**Value**

the R object automatically converted from julia object.

## Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
    ## doing initialization and automatic installation of Julia if necessary
    julia_setup(installJulia = TRUE)
    julia_eval("sqrt(2)")
}
```

---

julia_exists	<i>Check whether a julia object with the given name exists or not.</i>
--------------	--

---

## Description

julia\_exists returns whether a julia object with the given name exists or not.

## Usage

```
julia_exists(name)
```

## Arguments

name            the name of julia object you want to check.

## Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
    ## doing initialization and automatic installation of Julia if necessary
    julia_setup(installJulia = TRUE)
    julia_exists("sqrt")
}
```

---

julia_help	<i>Get help for a julia function.</i>
------------	---------------------------------------

---

## Description

julia\_help outputs the documentation of a julia function.

## Usage

```
julia_help(fname)
```

**Arguments**

fname            the name of julia function you want to get help with.

**Examples**

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_help("sqrt")
}
```

---

julia\_markdown\_setup    *Do setup for JuliaCall in RMarkdown documents and notebooks.*

---

**Description**

julia\_markdown\_setup does the initial setup for JuliaCall in RMarkdown document and RStudio notebooks. The function should be invoked automatically most of the case. It can also be called explicitly in RMarkdown documents or notebooks.

**Usage**

```
julia_markdown_setup(..., notebook = TRUE)
```

**Arguments**

...            The same arguments accepted by 'julia\_setup'.  
notebook       whether it is in RStudio notebook environment or not.

---

julia\_notebook\_setup    *(Deprecated) Do setup for julia chunks in RMarkdown notebooks.*

---

**Description**

julia\_notebook\_setup is deprecated, use julia\_markdown\_setup(notebook=TRUE) instead.

**Usage**

```
julia_notebook_setup(...)
```

**Arguments**

...            The same arguments accepted by 'julia\_setup'.

---

julia\_package            *Using julia packages.*

---

### Description

Using julia packages.

### Usage

```
julia_install_package(pkg_name_or_url)

julia_installed_package(pkg_name)

julia_install_package_if_needed(pkg_name)

julia_update_package(...)

julia_library(pkg_name)
```

### Arguments

pkg_name_or_url	the julia package name or url.
pkg_name	the julia package name.
...	you can provide none or one or multiple julia package names here.

### Value

`julia_installed_package` will return the version number of the julia package, "nothing" if the package is not installed.

### Examples

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_install_package("DataFrames")
  julia_installed_package("DataFrames")
  julia_install_package_if_needed("DataFrames")
  julia_update_package("DataFrames")
  julia_library("DataFrames")
}
```

---

julia_pkg_wrap	<i>Wrap julia functions and packages the easy way.</i>
----------------	--

---

**Description**

Wrap julia functions and packages the easy way.

**Usage**

```
julia_function(func_name, pkg_name = "Main", env = new.env(emptyenv()))
julia_pkg_import(pkg_name, func_list, env = new.env(parent = emptyenv()))
julia_pkg_hook(env, hook)
```

**Arguments**

func_name	the julia function name to be wrapped.
pkg_name	the julia package name to be wrapped.
env	the environment where the functions and packages are wrapped.
func_list	the list of julia function names to be wrapped.
hook	the function to be executed before the execution of wrapped functions.

**Examples**

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## do initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  julia_install_package_if_needed("Optim")
  opt <- julia_pkg_import("Optim",
                        func_list = c("optimize", "BFGS"))
  rosenbrock <- function(x) (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
  result <- opt$optimize(rosenbrock, rep(0,2), opt$BFGS())
  result
}
```

---

julia_setup	<i>Do initial setup for the JuliaCall package.</i>
-------------	--

---

**Description**

julia\_setup does the initial setup for the JuliaCall package. It setups automatic type conversion, Julia display systems, etc, and is necessary for every new R session to use the package. If not carried out manually, it will be invoked automatically before other julia\_xxx functions.

**Usage**

```
julia_setup(
  JULIA_HOME = NULL,
  verbose = TRUE,
  installJulia = FALSE,
  install = TRUE,
  force = FALSE,
  useRCall = TRUE,
  rebuild = FALSE,
  sysimage_path = NULL
)
```

**Arguments**

JULIA_HOME	the file folder which contains julia binary, if not set, JuliaCall will look at the global option JULIA_HOME, if the global option is not set, JuliaCall will then look at the environmental variable JULIA_HOME, if still not found, JuliaCall will try to use the julia in path.
verbose	whether to print out detailed information about julia_setup.
installJulia	whether to install julia automatically when julia is not found, whose default value is FALSE.
install	whether to execute installation script for dependent julia packages, whose default value is TRUE; but can be set to FALSE to save startup time when no installation of dependent julia packages is needed.
force	whether to force julia_setup to execute again.
useRCall	whether or not you want to use RCall.jl in julia, which is an amazing package to access R in julia.
rebuild	whether to rebuild RCall.jl, whose default value is FALSE to save startup time. If a new version of R is used, then this parameter needs to be set to TRUE.
sysimage_path	path to the precompiled custom sys image. Path can be either an absolute path or relative to the current directory.

**Value**

The julia interface, which is an environment with the necessary methods like command, source and things like that to communicate with julia.

**Examples**

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  julia <- julia_setup(installJulia = TRUE)
}
```

---

julia_source	<i>Source a julia source file.</i>
--------------	------------------------------------

---

**Description**

julia\_source sources a julia source file.

**Usage**

```
julia_source(file_name)
```

**Arguments**

file\_name      the name of julia source file.

---

plotsViewer	<i>Julia plots viewer in R.</i>
-------------	---------------------------------

---

**Description**

plotsViewer lets you view julia plots in R.

**Usage**

```
plotsViewer()
```

---

%>J%	<i>Language piper for julia language.</i>
------	---

---

**Description**

The experimental language piper for julia language.

**Usage**

```
obj %>J% func_call
```

**Arguments**

obj              the object to pass to the piper.  
func\_call        the impartial julia function call.

**Examples**

```
if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## julia_setup is quite time consuming
  ## doing initialization and automatic installation of Julia if necessary
  julia_setup(installJulia = TRUE)
  2 %>J% sqrt
  3 %>J% log(2)
}
```



# Index

`%>J%`, 15

`autowrap`, 2

`call`, 3

`eng_juliacall`, 4

`field`(`JuliaObjectFields`), 7

`field<-`(`JuliaObjectFields`), 7

`fields`(`JuliaObjectFields`), 7

`install_julia`, 4

`julia_assign`, 7

`julia_call`(`call`), 3

`julia_command`, 8

`julia_console`, 9

`julia_do.call`(`call`), 3

`julia_eval`, 9

`julia_exists`, 10

`julia_function`(`julia_pkg_wrap`), 13

`julia_help`, 10

`julia_install_package`(`julia_package`), 12

`julia_install_package_if_needed`(`julia_package`), 12

`julia_installed_package`(`julia_package`), 12

`julia_library`(`julia_package`), 12

`julia_markdown_setup`, 11

`julia_notebook_setup`, 11

`julia_package`, 12

`julia_pkg_hook`(`julia_pkg_wrap`), 13

`julia_pkg_import`(`julia_pkg_wrap`), 13

`julia_pkg_wrap`, 13

`julia_setup`, 13

`julia_source`, 15

`julia_update_package`(`julia_package`), 12

`JuliaCall`, 5

`JuliaObject`, 6

`JuliaObjectFields`, 7

`plotsViewer`, 15