

# Package: rstudioapi (via r-universe)

June 30, 2024

**Title** Safely Access the RStudio API

**Description** Access the RStudio API (if available) and provide informative error messages when it's not.

**Version** 0.16.0.9000

**Maintainer** Kevin Ushey <kevin@rstudio.com>

**License** MIT + file LICENSE

**URL** <https://rstudio.github.io/rstudioapi/>,  
<https://github.com/rstudio/rstudioapi>

**BugReports** <https://github.com/rstudio/rstudioapi/issues>

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**Suggests** testthat, knitr, rmarkdown, clipr, covr

**VignetteBuilder** knitr

**Encoding** UTF-8

**Repository** <https://rstudio.r-universe.dev>

**RemoteUrl** <https://github.com/rstudio/rstudioapi>

**RemoteRef** HEAD

**RemoteSha** a98549272c1b7a782e7578306b824f785410ff7a

## Contents

addTheme . . . . .	3
applyTheme . . . . .	4
askForPassword . . . . .	5
askForSecret . . . . .	5
bugReport . . . . .	6
build-tools . . . . .	6
callFun . . . . .	7
chunk-callbacks . . . . .	8
convertTheme . . . . .	8

createProjectTemplate . . . . .	9
dictionaries . . . . .	10
document_position . . . . .	11
document_range . . . . .	11
executeCommand . . . . .	12
file-dialogs . . . . .	13
filesPaneNavigate . . . . .	14
getActiveProject . . . . .	14
getDelegatedAzureToken . . . . .	15
getRStudioPackageDependencies . . . . .	15
getThemeInfo . . . . .	16
getThemes . . . . .	16
getVersion . . . . .	17
hasColorConsole . . . . .	17
hasFun . . . . .	18
highlightUi . . . . .	19
isAvailable . . . . .	20
isJob . . . . .	21
jobAdd . . . . .	21
jobAddOutput . . . . .	23
jobAddProgress . . . . .	23
jobGetState . . . . .	24
jobList . . . . .	24
jobRemove . . . . .	25
jobRunScript . . . . .	25
jobSetProgress . . . . .	26
jobSetState . . . . .	26
jobSetStatus . . . . .	27
launcherAvailable . . . . .	28
launcherConfig . . . . .	28
launcherContainer . . . . .	29
launcherControlJob . . . . .	29
launcherGetInfo . . . . .	30
launcherGetJob . . . . .	30
launcherGetJobs . . . . .	31
launcherHostMount . . . . .	31
launcherNfsMount . . . . .	32
launcherPlacementConstraint . . . . .	33
launcherResourceLimit . . . . .	33
launcherSubmitJob . . . . .	34
launcherSubmitR . . . . .	36
navigateToFile . . . . .	36
persistent-values . . . . .	37
previewRd . . . . .	38
previewSql . . . . .	39
primary_selection . . . . .	39
projects . . . . .	40
readPreference . . . . .	40

readRStudioPreference . . . . .	41
registerCommandCallback . . . . .	42
registerCommandStreamCallback . . . . .	43
removeTheme . . . . .	44
restartSession . . . . .	45
rstudio-documents . . . . .	45
rstudio-editors . . . . .	48
savePlotAsImage . . . . .	49
selections . . . . .	49
sendToConsole . . . . .	50
setGhostText . . . . .	50
showDialog . . . . .	51
showPrompt . . . . .	51
showQuestion . . . . .	52
sourceMarkers . . . . .	52
systemUsername . . . . .	53
terminalActivate . . . . .	54
terminalBuffer . . . . .	55
terminalBusy . . . . .	55
terminalClear . . . . .	56
terminalContext . . . . .	57
terminalCreate . . . . .	58
terminalExecute . . . . .	59
terminalExitCode . . . . .	60
terminalKill . . . . .	60
terminalList . . . . .	61
terminalRunning . . . . .	61
terminalSend . . . . .	62
terminalVisible . . . . .	63
translateLocalUrl . . . . .	63
unregisterCommandCallback . . . . .	64
updateDialog . . . . .	64
userIdentity . . . . .	65
versionInfo . . . . .	65
viewer . . . . .	66
writePreference . . . . .	68
writeRStudioPreference . . . . .	68

**Index****70**

addTheme

*Add a Custom Editor Theme***Description**

Adds a custom editor theme to RStudio and returns the name of the newly added theme.

**Usage**

```
addTheme(themePath, apply = FALSE, force = FALSE, globally = FALSE)
```

**Arguments**

themePath	A full or relative path or URL to an rstheme or tmtheme to be added.
apply	Whether to immediately apply the newly added theme. Setting this to TRUE has the same impact as running <code>{ rstudioapi::addTheme(&lt;themePath&gt;); rstudioapi::applyTheme(&lt;th&gt;}</code> . Default: FALSE.
force	Whether to force the operation and overwrite an existing file with the same name. Default: FALSE.
globally	Whether to install this theme for the current user or all users. If set to TRUE this will attempt to install the theme for all users, which may require administrator privileges. Default: FALSE.

**Note**

The addTheme function was introduced in RStudio 1.2.879.

---

applyTheme	<i>Apply an Editor Theme to RStudio</i>
------------	---

---

**Description**

Applies the specified editor theme to RStudio.

**Usage**

```
applyTheme(name)
```

**Arguments**

name	The unique name of the theme to apply.
------	--

**Note**

The applyTheme function was introduced in RStudio 1.2.879.

---

askForPassword	<i>Ask the user for a password interactively</i>
----------------	--

---

**Description**

Ask the user for a password interactively.

**Usage**

```
askForPassword(prompt = "Please enter your password")
```

**Arguments**

prompt            The prompt to be shown to the user.

**Details**

RStudio also sets the global `askpass` option to the `rstudioapi::askForPassword` function so that it can be invoked in a front-end independent manner.

**Note**

The `askForPassword` function was added in version 0.99.853 of RStudio.

**Examples**

```
## Not run:  
rstudioapi::askForPassword("Please enter your password")  
  
## End(Not run)
```

---

askForSecret	<i>Prompt user for secret</i>
--------------	-------------------------------

---

**Description**

Request a secret from the user. If the `keyring` package is installed, it will be used to cache requested secrets.

**Usage**

```
askForSecret(  
  name,  
  message = paste(name, ":", sep = ""),  
  title = paste(name, "Secret")  
)
```

**Arguments**

name	The name of the secret.
message	A character vector with the contents to display in the main dialog area.
title	The title to display in the dialog box.

**Note**

The askForSecret function was added in version 1.1.419 of RStudio.

---

bugReport	<i>File an RStudio Bug Report</i>
-----------	-----------------------------------

---

**Description**

A utility function to assist with the filing of an RStudio bug report. This function will pre-populate a template with information useful in understanding your reported bug.

**Usage**

```
bugReport()
```

---

build-tools	<i>Build Tools</i>
-------------	--------------------

---

**Description**

Check, install, and use build tools as required.

**Usage**

```
buildToolsCheck()
```

```
buildToolsInstall(action)
```

```
buildToolsExec(expr)
```

**Arguments**

action	The action (as a string) being taken that will require installation of build tools.
expr	An R expression (unquoted) to be executed with build tools available and on the PATH.

## Details

These functions are intended to be used together – one should first check whether build tools are available, and when not, prompt for installation. For example:

```
compile_model <- function(...) {  
  if (rstudioapi::isAvailable()) {  
    if (!rstudioapi::buildToolsCheck())  
      rstudioapi::buildToolsInstall("Model compilation")  
  
    rstudioapi::buildToolsExec({  
      # code requiring build tools here  
    })  
  }  
}
```

The action parameter is used to communicate (with a prompt) the operation being performed that requires build tool installation. Setting it to NULL or the empty string will suppress that prompt.

## Note

The `buildToolsCheck()`, `buildToolsInstall()`, and `buildToolsExec()` functions were added with version 1.2.962 of RStudio.

---

callFun

*Call an RStudio API function*

---

## Description

This function will return an error if RStudio is not running, or the function is not available. If you want to fall back to different behavior, use [hasFun](#).

## Usage

```
callFun(fname, ...)
```

## Arguments

fname	name of the RStudio function to call.
...	Other arguments passed on to the function

## Examples

```
if (rstudioapi::isAvailable()) {  
  rstudioapi::callFun("versionInfo")  
}
```

---

chunk-callbacks	<i>Register and Unregister a Chunk Callback</i>
-----------------	---

---

### Description

Register a callback function to be executed after a chunk within an R Markdown document is run.

### Usage

```
registerChunkCallback(callback)
```

```
unregisterChunkCallback(id = NULL)
```

### Arguments

callback	A callback function. See <b>Chunk Callbacks</b> for more details.
id	A unique identifier.

### Value

For `registerChunkCallback()`, a unique identifier. That identifier can be passed to `unregisterChunkCallback()` to de-register a previously-registered callback.

### Chunk Callbacks

The callback argument should be a function accepting two parameters:

- chunkName: The chunk label,
- chunkCode: The code within the chunk.

The function should return an R list of HTML outputs, to be displayed after that chunk has been executed.

---

convertTheme	<i>Convert a tmTheme to an RStudio Theme</i>
--------------	--

---

### Description

Converts a `tmTheme` to an `rstheme` and optionally adds and applies it to RStudio and returns the name of the theme.



**Usage**

```
convertTheme(
  themePath,
  add = TRUE,
  outputLocation = NULL,
  apply = FALSE,
  force = FALSE,
  globally = FALSE
)
```

**Arguments**

themePath	A full or relative path to the tmTheme file to be converted.
add	Whether to add the newly converted theme to RStudio. Setting this to true will have the same impact as running { rstudioapi::convertTheme(<themePath>, outputLocation = <convertedThemePath>); rstudioapi::addTheme(<convertedThemePath>) }. Default: TRUE.
outputLocation	A full or relative path where a copy of the converted theme will be saved. If this value is NULL, no copy will be saved. Default: NULL.
apply	Whether to immediately apply the newly added theme. This parameter cannot be set to TRUE if add is set to FALSE. Setting this and add to TRUE has the same impact as running { rstudioapi::convertTheme(<themePath>, outputLocation = <convertedThemePath>); rstudioapi::addTheme(<convertedThemePath>); rstudioapi::applyTheme(<themeName>) }. Default: FALSE.
force	Whether to force the operation and overwrite an existing file with the same name. Default: FALSE.
globally	Whether to install this theme for the current user or all users. If set to TRUE this will attempt to install the theme for all users, which may require administrator privileges. Only applies when add is TRUE. Default: FALSE.

**Note**

The convertTheme function was introduced in RStudio 1.2.879.

---

createProjectTemplate *Create a Project Template*

---

**Description**

Create a project template. See [https://rstudio.github.io/rstudio-extensions/rstudio\\_project\\_templates.html](https://rstudio.github.io/rstudio-extensions/rstudio_project_templates.html) for more information.

**Usage**

```
createProjectTemplate(
  package = ".",
  binding,
  title,
  subtitle = paste("Create a new", title),
  caption = paste("Create", title),
  icon = NULL,
  open_files = NULL,
  overwrite = FALSE,
  edit = TRUE
)
```

**Arguments**

package	The path to an package sources.
binding	The skeleton function to associate with this project template. This is the name of the function that will be used to initialize the project.
title	The title to be shown within the <b>New Project...</b> wizard.
subtitle	(optional) The subtitle to be shown within the <b>New Project...</b> wizard.
caption	(optional) The caption to be shown on the landing page for this template.
icon	(optional) The path to an icon, on disk, to be used in the dialog. Must be an .png of size less than 64KB.
open_files	(optional) Files that should be opened by RStudio when the project is generated. Shell-style globs can be used to indicate when multiple files matching some pattern should be opened – for example, OpenFiles: R/*.R would indicate that RStudio should open all .R files within the R folder of the generated project.
overwrite	Boolean; overwrite a pre-existing template file if one exists?
edit	Boolean; open the file for editing after creation?

---

dictionaries

*Interact with RStudio's Dictionaries*


---

**Description**

Interact with the **hunspell** dictionaries used by RStudio for spell checking.

**Usage**

```
dictionariesPath()

userDictionariesPath()
```

**Details**

dictionariesPath() gives a path to the dictionaries installed and distributed with RStudio.

userDictionariesPath() gives the path where users can provide their own custom hunspell dictionaries.

**Note**

The dictionariesPath() and userDictionariesPath() functions were introduced with RStudio 1.2.1202.

---

document_position	<i>Create a Document Position</i>
-------------------	-----------------------------------

---

**Description**

Creates a document\_position, which can be used to indicate e.g. the row + column location of the cursor in a document.

**Usage**

```
document_position(row, column)
```

```
is.document_position(x)
```

```
as.document_position(x)
```

**Arguments**

row	The row (using 1-based indexing).
column	The column (using 1-based indexing).
x	An object coercable to document_position.

---

document_range	<i>Create a Range</i>
----------------	-----------------------

---

**Description**

A document\_range is a pair of [document\\_position](#) objects, with each position indicating the start and end of the range, respectively.

**Usage**

```
document_range(start, end = NULL)
```

```
is.document_range(x)
```

```
as.document_range(x)
```

**Arguments**

start	A <a href="#">document_position</a> indicating the start of the range.
end	A <a href="#">document_position</a> indicating the end of the range.
x	An object coercable to <code>document_range</code> .

**Value**

An list with class `document_range` and fields:

start:	The start position.
end:	The end position.

---

executeCommand

*Execute Command*

---

**Description**

Executes an arbitrary RStudio command.

**Usage**

```
executeCommand(commandId, quiet = FALSE)
```

**Arguments**

commandId	The ID of the command to execute.
quiet	Whether to show an error if the command does not exist.

**Details**

Most menu commands and many buttons in RStudio can be invoked from the API using this method. The `quiet` command governs the behavior of the function when the command does not exist. By default, an error is shown if you attempt to invoke a non-existent command. You should set this to `TRUE` when invoking a command that may not be available if you don't want your users to see an error.

The command is run asynchronously, so no status is returned.

See the [RStudio Server Professional Administration Guide](#) for a list of supported command IDs.

**Note**

The `executeCommand` function was introduced in RStudio 1.2.1261.

**See Also**

[registerCommandCallback](#) to be notified of command executions.

---

file-dialogs	<i>Select a file / folder</i>
--------------	-------------------------------

---

**Description**

Prompt the user for the path to a file or folder, using the system file dialogs with RStudio Desktop, and RStudio's own dialogs with RStudio Server.

**Usage**

```
selectFile(  
  caption = "Select File",  
  label = "Select",  
  path = getActiveProject(),  
  filter = "All Files (*)",  
  existing = TRUE  
)  
  
selectDirectory(  
  caption = "Select Directory",  
  label = "Select",  
  path = getActiveProject()  
)
```

**Arguments**

<code>caption</code>	The window title.
<code>label</code>	The label to use for the 'Accept' / 'OK' button.
<code>path</code>	The initial working directory, from which the file dialog should begin browsing. Defaults to the current RStudio project directory.
<code>filter</code>	A glob filter, to be used when attempting to open a file with a particular extension. For example, to scope the dialog to R files, one could use <code>R Files (*.R)</code> here.
<code>existing</code>	Boolean; should the file dialog limit itself to existing files on the filesystem, or allow the user to select the path to a new file?

**Details**

When the selected file resolves within the user's home directory, RStudio will return an aliased path – that is, prefixed with `~/`.

**Note**

The `selectFile` and `selectDirectory` functions were added in version 1.1.287 of RStudio.

---

<code>filesPaneNavigate</code>	<i>Navigate to a Directory in the Files Pane</i>
--------------------------------	--

---

**Description**

Navigate to a directory in the Files pane. The contents of that directory will be listed and shown in the Files pane.

**Usage**

```
filesPaneNavigate(path)
```

**Arguments**

<code>path</code>	The filesystem path to be shown.
-------------------	----------------------------------

---

<code>getActiveProject</code>	<i>Retrieve path to active RStudio project</i>
-------------------------------	--

---

**Description**

Get the path to the active RStudio project (if any). If the path contains non-ASCII characters, it will be UTF-8 encoded.

**Usage**

```
getActiveProject()
```

**Value**

The path to the current project, or NULL if no project is currently open.

**Note**

The `getActiveProject` function was added in version 0.99.854 of RStudio.

---

```
getDelegatedAzureToken
```

*OAuth2 Tokens for Delegated Azure Resources*

---

### Description

When Workbench is using Azure Active Directory for sign-in, this function can return an OAuth2 token for a service Workbench users have delegated access to. This requires configuring delegated permissions in Azure itself.

### Usage

```
getDelegatedAzureToken(resource)
```

### Arguments

`resource`            The name of an Azure resource or service, normally a URL.

### Examples

```
## Not run:  
getDelegatedAzureToken("https://storage.azure.com")  
  
## End(Not run)
```

---

```
getRStudioPackageDependencies
```

*Get RStudio Package Dependencies*

---

### Description

Gets a list of the all the R packages that RStudio depends on in some way.

### Usage

```
getRStudioPackageDependencies()
```

### Details

The data frame of package dependencies contains the following columns:

**name** The name of the R package.

**version** The required minimum version of the R package.

**location** Where RStudio expects the package to be, cran for a CRAN-like repository or embedded for development packages embedded in RStudio itself.

**source** Whether the package should be installed from source.

**Value**

A data frame containing a row per R package.

**Note**

The getRStudioPackageDependencies function was introduced in RStudio 1.3.525.

---

getThemeInfo	<i>Retrieve Themes</i>
--------------	------------------------

---

**Description**

Retrieves a list with information about the current color theme used by RStudio.

**Usage**

```
getThemeInfo()
```

**Details**

A list is returned with the following elements:

**editor** The name of the current editor theme, such as Textmate.

**global** The name of the current global theme. One of Modern, Classic, or Sky.

**dark** TRUE if the editor theme is dark, FALSE otherwise.

**foreground** The current editor theme's default text foreground color, formatted as a CSS-compatible color string, such as rgb(1, 22, 39). Supported since RStudio 1.2.1214.

**background** The current editor theme's default text background color, formatted as a CSS-compatible color string. Supported since RStudio 1.2.1214.

---

getThemes	<i>Get Theme List</i>
-----------	-----------------------

---

**Description**

Retrieves a list of the names of all the editor themes installed for RStudio.

**Usage**

```
getThemes()
```

**Note**

The getThemes function was introduced in RStudio 1.2.879.



---

getVersion	<i>Return the current version of the RStudio API</i>
------------	--

---

**Description**

Return the current version of the RStudio API

**Usage**

```
getVersion()
```

**Value**

A `numeric_version` which you can compare to a string and get correct results.

**Examples**

```
## Not run:  
if (rstudioapi::getVersion() < "0.98.100") {  
  message("Your version of RStudio is quite old")  
}  
  
## End(Not run)
```

---

hasColorConsole	<i>Check if console supports ANSI color escapes.</i>
-----------------	--

---

**Description**

Check if the RStudio console supports ANSI color escapes.

**Usage**

```
hasColorConsole()
```

**Value**

TRUE if ANSI color escapes are supported; FALSE otherwise.

**Note**

The `hasColorConsole` function was added in version 1.1.216 of RStudio.

## Examples

```
## Not run:
if (rstudioapi::hasColorConsole()) {
  message("RStudio console supports ANSI color sequences.")
}

## End(Not run)
```

---

hasFun	<i>Exists/get for RStudio functions</i>
--------	---

---

## Description

These are specialized versions of `get` and `exists` that look in the `rstudio` package namespace. If RStudio is not running, `hasFun` will return `FALSE`.

## Usage

```
hasFun(name, version_needed = NULL, ...)
findFun(name, version_needed = NULL, ...)
```

## Arguments

<code>name</code>	name of object to look for
<code>version_needed</code>	An optional version specification. If supplied, ensures that RStudio is at least that version. This is useful if function behavior has changed over time.
<code>...</code>	other arguments passed on to <code>exists</code> and <code>get</code>

## Examples

```
rstudioapi::hasFun("viewer")
```

---

`highlightUi`*Highlight UI Elements within the RStudio IDE*

---

## Description

This function can be used to highlight UI elements within the RStudio IDE. UI elements can be selected using query selectors; most commonly, one should choose to highlight elements based on their IDs when available.

## Usage

```
highlightUi(queries)
```

## Arguments

`queries` A list of "query" objects. Each query should be a list with entries "query" and "parent". See **Queries** for more details.

## Details

The tool at:

```
Help -> Diagnostics -> Show DOM Elements
```

can be useful for identifying the classes and IDs assigned to the different elements within RStudio.

## Queries

Elements are selected using the same queries as through the web `querySelectorAll()` API. See <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll> for more details.

For example, to highlight the Save icon within the Source pane, one might use:

```
rstudioapi::highlightUi("#rstudio_tb_savesourcedoc")
```

In some cases, multiple UI elements need to be highlighted – e.g. if you want to highlight both a menu button, and a menu item within the menu displayed after the button is pressed. We'll use the Environment Pane's Import Dataset button as an example. To highlight the From Text (readr) command, you might use:

```
rstudioapi::highlightUi( list(
  list(query = "#rstudio_mb_import_dataset", parent = 0L), list(query =
  "#rstudio_label_from_text_readr_command", parent = 1L) ) )
```

## Note

The `highlightUi` function was introduced in RStudio 1.3.658.

**Examples**

```
## Not run: rstudioapi::highlightUi("#rstudio_workbench_panel_git")

# clear current highlights
## Not run: rstudioapi::highlightUi("")

# highlight within an RMD
## Not run: rstudioapi::highlightUi(".rstudio_chunk_setup .rstudio_run_chunk")

# Optionally provide a callback adjacent to
# the queries that will be executed when the
# highlighted element is clicked on.
## Not run: rstudioapi::highlightUi(
  list(
    list(
      query="#rstudio_workbench_panel_git",
      callback="rstudioapi::highlightUi('')"
    )
  )
)
## End(Not run)
```

---

isAvailable

*Check if RStudio is running*


---

**Description**

Check if RStudio is running.

**Usage**

```
isAvailable(version_needed = NULL, child_ok = FALSE)
```

```
verifyAvailable(version_needed = NULL)
```

**Arguments**

`version_needed` An optional version specification. If supplied, ensures that RStudio is at least that version.

`child_ok` Boolean; check if the current R process is a child process of the main RStudio session? This can be useful for e.g. RStudio Jobs, where you'd like to communicate back with the main R session from a child process through `rstudioapi`.

**Value**

`isAvailable` a boolean; `verifyAvailable` an error message if RStudio is not running

**Examples**

```
rstudioapi::isAvailable()  
## Not run: rstudioapi::verifyAvailable()
```

---

*isJob**Detect RStudio Jobs*

---

**Description**

Use this function to detect whether RStudio is running an R "job". These jobs are normally used for actions taken in the Jobs tab, as well as within the R build pane.

**Usage**

```
isJob()
```

```
isBackgroundJob()
```

```
isWorkbenchJob()
```

**Details**

`isWorkbenchJob()` is used to detect scripts which have been launched as Workbench jobs, and is only available in RStudio Workbench 2024.04 or newer. These jobs use the RStudio Launcher to run R scripts on remote clusters, as opposed to `isBackgroundJob()`, which is used to detect background jobs which are run on the local machine.

This function is primarily intended to be used by package authors, who need to customize the behavior of their methods when run within an RStudio job.

**Value**

Boolean; TRUE if this is an RStudio job.

---

*jobAdd**Add a Job*

---

**Description**

Inform RStudio's Background Jobs pane that a job has been added.

**Usage**

```

jobAdd(
  name,
  status = "",
  progressUnits = 0L,
  actions = NULL,
  running = FALSE,
  autoRemove = TRUE,
  show = TRUE
)

```

**Arguments**

name	The background job's name.
status	The initial status text for the job; optional.
progressUnits	The integer number of units of work in the job; for example, 100L if the job's progress is expressed in percentages. Use 0L if the number of units of work is unknown.
actions	A list of actions that can be performed on the job (see Actions).
running	Whether the job is currently running.
autoRemove	Whether to remove the job from the Background Jobs pane when it's complete.
show	Whether to show the job in the Jobs pane.

**Value**

An ID representing the newly added job, used as a handle to provide further updates of the job's status.

**Actions**

The actions parameter is a named list of functions that the user can invoke on the job; for example: `actions = list(stop = function(id) { ... })`. The function will be passed a parameter named `id` with the job ID that invoked it.

There are three special action names:

**stop** If there is an action named `stop`, then the job will have a Stop button in in the Jobs pane, and pressing that button will invoke the `stop` action.

**info** If there is an action named `info`, then the job will have an informational link in the Background Jobs pane rather than an output display, and clicking the link will invoke the `info` action.

**replay** If there is an action named `replay`, then the job will have a Replay button that displays when the job has finished running. Clicking the button will invoke the `replay` action.

**See Also**

Other jobs: [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobAddOutput	<i>Add Background Job Output</i>
--------------	----------------------------------

---

**Description**

Adds text output to a background job.

**Usage**

```
jobAddOutput(job, output, error = FALSE)
```

**Arguments**

job	The ID of the job that has emitted text.
output	The text output emitted by the job.
error	Whether the output represents an error.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobAddProgress	<i>Add Background Job Progress</i>
----------------	------------------------------------

---

**Description**

Adds incremental progress units to a background job.

**Usage**

```
jobAddProgress(job, units)
```

**Arguments**

job	The ID of the job to update progress for.
units	The integer number of new progress units completed.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobGetState	<i>Get Background Job State</i>
-------------	---------------------------------

---

**Description**

Get Background Job State

**Usage**

```
jobGetState(job)
```

**Arguments**

job	The ID of the job.
-----	--------------------

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobList	<i>List Background Jobs</i>
---------	-----------------------------

---

**Description**

List any registered background jobs.

**Usage**

```
jobList()
```

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)



---

jobRemove	<i>Remove a Background Job</i>
-----------	--------------------------------

---

**Description**

Remove a background job from RStudio's Background Jobs pane.

**Usage**

```
jobRemove(job)
```

**Arguments**

job	The ID of the job to remove.
-----	------------------------------

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobRunScript	<i>Run R Script As Background Job</i>
--------------	---------------------------------------

---

**Description**

Starts an R script as a background job.

**Usage**

```
jobRunScript(  
  path,  
  name = NULL,  
  encoding = "unknown",  
  workingDir = NULL,  
  importEnv = FALSE,  
  exportEnv = ""  
)
```

**Arguments**

path	The path to the R script to be run.
name	A name for the background job. When NULL (the default), the filename of the script is used as the job name.
encoding	The text encoding of the script, if known.

workingDir	The working directory in which to run the job. When NULL (the default), the parent directory of the R script is used.
importEnv	Whether to import the global environment into the job.
exportEnv	The name of the environment in which to export the R objects created by the job. Use "" (the default) to skip export, "R_GlobalEnv" to export to the global environment, or the name of an environment object to create an object with that name.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobSetProgress	<i>Set Background Job Progress</i>
----------------	------------------------------------

---

**Description**

Updates the progress for a background job.

**Usage**

```
jobSetProgress(job, units)
```

**Arguments**

job	The ID of the job to set progress for.
units	The integer number of total units of work completed so far.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetState\(\)](#), [jobSetStatus\(\)](#)

---

jobSetState	<i>Set Background Job State</i>
-------------	---------------------------------

---

**Description**

Changes the state of a background job.

**Usage**

```
jobSetState(
  job,
  state = c("idle", "running", "succeeded", "cancelled", "failed")
)
```

**Arguments**

job	The ID of the job on which to change state.
state	The new job state.

**States**

The following states are supported:

**idle** The job is waiting to run.

**running** The job is actively running.

**succeeded** The job has finished successfully.

**cancelled** The job was cancelled.

**failed** The job finished but did not succeed.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetStatus\(\)](#)

---

jobSetStatus	<i>Set Background Job Status</i>
--------------	----------------------------------

---

**Description**

Update a background job's informational status text.

**Usage**

```
jobSetStatus(job, status)
```

**Arguments**

job	The ID of the job to update.
status	Text describing job's new status.

**See Also**

Other jobs: [jobAdd\(\)](#), [jobAddOutput\(\)](#), [jobAddProgress\(\)](#), [jobGetState\(\)](#), [jobList\(\)](#), [jobRemove\(\)](#), [jobRunScript\(\)](#), [jobSetProgress\(\)](#), [jobSetState\(\)](#)

---

launcherAvailable	<i>Check if Workbench Launcher is Available</i>
-------------------	---

---

**Description**

Check if the Workbench launcher is available and configured to support Workbench jobs; that is, jobs normally launched by the user through the RStudio IDE's user interface.

**Usage**

```
launcherAvailable()
```

**See Also**

Other job-launcher functionality: [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherConfig	<i>Define a Workbench Launcher Configuration</i>
----------------	--

---

**Description**

Define a Workbench launcher configuration, suitable for use with the config argument to [launcherSubmitJob\(\)](#).

**Usage**

```
launcherConfig(name, value = NULL)
```

**Arguments**

name	The name of the launcher configuration.
value	The configuration value. Must either be an integer, float, or string.

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherContainer      *Define a Workbench Launcher Container*

---

### Description

Define a launcher container, suitable for use with the container argument to [launcherSubmitJob\(\)](#).

### Usage

```
launcherContainer(image, runAsUserId = NULL, runAsGroupId = NULL)
```

### Arguments

image	The container image to use.
runAsUserId	The user id to run as within the container. Defaults to the container-specified user.
runAsGroupId	The group id to run as within the container. Defaults to the container-specified group.

### See Also

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherControlJob      *Interact with (Control) a Workbench Job*

---

### Description

Interact with a Workbench job.

### Usage

```
launcherControlJob(
  jobId,
  operation = c("suspend", "resume", "stop", "kill", "cancel")
)
```

### Arguments

jobId	The job id.
operation	The operation to execute. The operation should be one of <code>c("suspend", "resume", "stop", "kill", "cancel")</code> . Note that different launcher plugins support different subsets of these operations – consult your launcher plugin documentation to see which operations are supported.

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherGetInfo	<i>Retrieve Workbench Launcher Information</i>
-----------------	--

---

**Description**

Retrieve information about the Workbench launcher, as well as the different clusters that the launcher has been configured to use.

**Usage**

```
launcherGetInfo()
```

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherGetJob	<i>Retrieve Workbench Job Information</i>
----------------	---

---

**Description**

Retrieve information on a Workbench job with id `jobId`.

**Usage**

```
launcherGetJob(jobId)
```

**Arguments**

<code>jobId</code>	The id of a Workbench job.
--------------------	----------------------------

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherGetJobs      *Retrieve Workbench Job Information*

---

### Description

Retrieve information on Workbench jobs.

### Usage

```
launcherGetJobs(
  statuses = NULL,
  fields = NULL,
  tags = NULL,
  includeSessions = FALSE
)
```

### Arguments

statuses	Return only jobs whose status matches one of statuses. Valid statuses are: Pending, Running, Suspended, Failed, Finished, Killed, Canceled. When NULL, all jobs are returned.
fields	Return a subset of fields associated with each job object. When NULL, all fields associated with a particular job are returned.
tags	An optional set of tags. Only jobs that have been assigned one of these requested tags will be returned.
includeSessions	Boolean; include jobs which are also operating as RStudio R sessions?

### See Also

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherHostMount      *Define a Workbench Launcher Host Mount*

---

### Description

Define a launcher host mount, suitable for use with the mounts argument to [launcherSubmitJob\(\)](#). This can be used to mount a path from the host into the generated container.

### Usage

```
launcherHostMount(path, mountPath, readOnly = TRUE)
```

**Arguments**

path	The host path to be mounted.
mountPath	The destination path for the mount in the container.
readOnly	Boolean; should the path be mounted read-only?

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherNfsMount      *Define a Workbench Launcher NFS Mount*

---

**Description**

Define a launcher NFS mount, suitable for use with the mounts argument to [launcherSubmitJob\(\)](#). This can be used to mount a path from a networked filesystem into a newly generated container.

**Usage**

```
launcherNfsMount(host, path, mountPath, readOnly = TRUE)
```

**Arguments**

host	The host name, or IP address, of the NFS server.
path	The NFS path to be mounted.
mountPath	The destination path for the mount in the container.
readOnly	Boolean; should the path be mounted read-only?

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)



---

 launcherPlacementConstraint

*Define a Workbench Launcher Placement Constraint*


---

### Description

Define a launcher placement constraint, suitable for use with the placementConstraints argument to [launcherSubmitJob\(\)](#).

### Usage

```
launcherPlacementConstraint(name, value = NULL)
```

### Arguments

name	The name of this placement constraint.
value	The value of the constraint. A job will only be placed on a requested node if the requested placement constraint is present.

### See Also

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

 launcherResourceLimit *Define a Workbench Launcher Resource Limit*


---

### Description

Define a launcher resource limit, suitable for use with the resourceLimits argument to [launcherSubmitJob\(\)](#).

### Usage

```
launcherResourceLimit(type, value)
```

### Arguments

type	The resource limit type. Must be one of cpuCount, cpuFrequency, cpuSet, cpuTime, memory, memorySwap. Different launcher plugins may support different subsets of these resource limit types; please consult the plugin documentation to learn which limits are supported.
value	The formatted value of the requested limit.

**See Also**

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherSubmitJob\(\)](#), [launcherSubmitR\(\)](#)

---

launcherSubmitJob      *Submit a Workbench Job*

---

**Description**

Submit a Workbench job. See <https://docs.posit.co/job-launcher/latest/index.html> for more information.

**Usage**

```
launcherSubmitJob(
  name,
  cluster = "Local",
  tags = NULL,
  command = NULL,
  exe = NULL,
  args = NULL,
  environment = NULL,
  stdin = NULL,
  stdoutFile = NULL,
  stderrFile = NULL,
  workingDirectory = NULL,
  host = NULL,
  container = NULL,
  exposedPorts = NULL,
  mounts = NULL,
  placementConstraints = NULL,
  resourceLimits = NULL,
  queues = NULL,
  config = NULL,
  user = Sys.getenv("USER"),
  applyConfigSettings = TRUE
)
```

**Arguments**

name	A descriptive name to assign to the job.
cluster	The name of the cluster this job should be submitted to.
tags	A set of user-defined tags, used for searching and querying jobs.
command	The command to run within the job. This is executed via the system shell. Only one of command or exe should be specified.

exe	The (fully pathed) executable to run within the job. Only one of command or exe should be specified.
args	An array of arguments to pass to the command / executable.
environment	A list of environment variables to be set for processes launched with this job.
stdin	Data to be written to stdin when the job process is launched.
stdoutFile	The file used for the job's generated standard output. Not all launcher plugins support this parameter.
stderrFile	The file used for the job's generated standard error. Not all launcher plugins support this parameter.
workingDirectory	The working directory to be used by the command / executable associated with this job.
host	The host that the job is running on, or the desired host during job submission.
container	The container to be used for launched jobs.
exposedPorts	The ports that are exposed by services running on a container. Only applicable to systems that support containers.
mounts	A list of mount points. See <a href="#">launcherHostMount()</a> and <a href="#">launcherNfsMount()</a> for more information.
placementConstraints	A list of placement constraints. See <a href="#">launcherPlacementConstraint()</a> for more information.
resourceLimits	A list of resource limits. See <a href="#">launcherResourceLimit()</a> for more information.
queues	A list of available submission queues for the cluster. Only applicable to batch systems like LSF.
config	A list of cluster-specific configuration options. See <a href="#">launcherConfig()</a> for more information.
user	The user-name of the job owner.
applyConfigSettings	Apply server-configured mounts, exposedPorts, and environment, in addition to any specified in this call.

### See Also

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitR\(\)](#)

---

launcherSubmitR	<i>Execute an R Script as a Workbench Job</i>
-----------------	---

---

### Description

Convenience function for running an R script as a Workbench job using whichever R is found on the path in the Workbench launcher cluster.

### Usage

```
launcherSubmitR(script, cluster = "Local", container = NULL)
```

### Arguments

script	Fully qualified path of R script. Must be a path that is available in the job container (if using containerized job cluster such as Kubernetes).
cluster	The name of the cluster this job should be submitted to.
container	The container to be used for launched jobs.

### Details

See [launcherSubmitJob\(\)](#) for running jobs with full control over command, environment, and so forth.

### See Also

Other job-launcher functionality: [launcherAvailable\(\)](#), [launcherConfig\(\)](#), [launcherContainer\(\)](#), [launcherControlJob\(\)](#), [launcherGetInfo\(\)](#), [launcherGetJob\(\)](#), [launcherGetJobs\(\)](#), [launcherHostMount\(\)](#), [launcherNfsMount\(\)](#), [launcherPlacementConstraint\(\)](#), [launcherResourceLimit\(\)](#), [launcherSubmitJob\(\)](#)

---

navigateToFile	<i>Navigate to file</i>
----------------	-------------------------

---

### Description

Open a file in RStudio, optionally at a specified location.

### Usage

```
navigateToFile(
  file = character(0),
  line = -1L,
  column = -1L,
  moveCursor = TRUE
)
```

**Arguments**

file	The file to be opened.
line	The line number where the cursor should be placed. When -1L (the default), the cursor will not be moved.
column	The column number where the cursor should be placed. When -1L (the default), the cursor will not be moved.
moveCursor	Boolean; should the cursor be moved to the requested (line, column) position? Set this to FALSE to preserve the existing cursor position in the document.

**Details**

The `navigateToFile` opens a file in RStudio. If the file is already open, its tab or window is activated.

Once the file is open, the cursor is moved to the specified location. If the `file` argument is empty (the default), then the file is the file currently in view if one exists. If the `line` and `column` arguments are both equal to -1L (the default), then the cursor position in the document that is opened will be preserved. Alternatively, `moveCursor` can be set to FALSE to preserve the cursor position.

Note that if your intent is to navigate to a particular function within a file, you can also cause RStudio to navigate there by invoking [View](#) on the function, which has the advantage of falling back on deparsing if the file is not available.

**Note**

The `navigateToFile` function was added in version 0.99.719 of RStudio.

---

persistent-values      *Persistent keys and values*

---

**Description**

Store persistent keys and values. Storage is per-project; if there is no project currently active, then a global store is used.

**Usage**

```
setPersistentValue(name, value)
```

```
getPersistentValue(name)
```

**Arguments**

name	The key name.
value	The key value.

**Value**

The stored value as a character vector (NULL if no value of the specified name is available).

**Note**

The `setPersistentValue` and `getPersistentValue` functions were added in version 1.1.57 of RStudio.

---

previewRd	<i>Preview an Rd topic in the Help pane</i>
-----------	---

---

**Description**

Preview an Rd topic in the Help pane.

**Usage**

```
previewRd(rdFile)
```

**Arguments**

`rdFile`            The path to an .Rd file.

**Note**

The `previewRd` function was added in version 0.98.191 of RStudio.

**Examples**

```
## Not run:  
rstudioapi::previewRd("~/MyPackage/man/foo.Rd")  
  
## End(Not run)
```

---

previewSql	<i>Preview SQL statement</i>
------------	------------------------------

---

**Description**

Makes use of 'DBI' and dbGetQuery() to preview a SQL statement for a given 'DBI' connection.

**Usage**

```
previewSql(conn, statement, ...)
```

**Arguments**

conn	The 'DBI' connection to be used to execute this statement.
statement	The SQL statement to execute. Either a path to a file containing a SQL statement or the SQL statement itself.
...	Additional arguments to be used in dbGetQuery().

**Note**

The previewSql function was introduced in RStudio 1.2.600

---

primary_selection	<i>Extract the Primary Selection</i>
-------------------	--------------------------------------

---

**Description**

By default, functions returning a document context will return a list of selections, including both the 'primary' selection and also 'other' selections (e.g. to handle the case where a user might have multiple cursors active). Use primary\_selection() to extract the primary selection.

**Usage**

```
primary_selection(x, ...)
```

**Arguments**

x	A document context, or a selection.
...	Optional arguments (currently ignored).

---

projects	<i>Open a project in RStudio</i>
----------	----------------------------------

---

**Description**

Initialize and open RStudio projects.

**Usage**

```
openProject(path = NULL, newSession = FALSE)
```

```
initializeProject(path = getwd())
```

**Arguments**

path	Either the path to an existing .Rproj file, or a path to a directory in which a new project should be initialized and opened.
newSession	Boolean; should the project be opened in a new session, or should the current RStudio session switch to that project? Note that TRUE values are only supported with RStudio Desktop and RStudio Server Pro.

**Details**

Calling `openProject()` without arguments effectively re-opens the currently open project in RStudio. When switching projects, users will be prompted to save any unsaved files; alternatively, you can explicitly save any open documents using `documentSaveAll()`.

**Note**

The `openProject` and `initializeProject` functions were added in version 1.1.287 of RStudio.

---

readPreference	<i>Read Preference</i>
----------------	------------------------

---

**Description**

Reads a user preference, useful to remember preferences across different R sessions for the same user.

**Usage**

```
readPreference(name, default)
```



**Arguments**

name	The name of the preference.
default	The default value to use when the preference is not available.

**Details**

User preferences can have arbitrary names and values. You must write the preference with [writePreference](#) before it can be read (otherwise its default value will be returned).

**Note**

The readPreference function was added in version 1.1.67 of RStudio.

**See Also**

[readRStudioPreference](#), which reads RStudio IDE preferences.

---

readRStudioPreference *Read RStudio Preference*

---

**Description**

Reads an internal RStudio IDE preference for the current user.

**Usage**

```
readRStudioPreference(name, default)
```

**Arguments**

name	The name of the preference.
default	The default value of the preference, returned if the preference is not found.

**Details**

RStudio IDE internal preferences include the values displayed in RStudio's Global Options dialog as well as a number of additional settings.

**Note**

The readRStudioPreference function was added in version 1.3.387 of RStudio.

**See Also**

[readPreference](#), which can be used to read arbitrary user (non-RStudio) preferences set with [writePreference](#).

`link{writeRStudioPreference}`, which can be used to write internal RStudio IDE preferences.

## Examples

```
## Not run:
# Get indentation settings
spaces <- rstudioapi::readRStudioPreference("num_spaces_for_tab", FALSE)
message("Using ", spaces, " per tab.")

## End(Not run)
```

---

registerCommandCallback

*Register Command Callback*

---

## Description

Registers a callback to be executed when an RStudio command is invoked.

## Usage

```
registerCommandCallback(commandId, callback)
```

## Arguments

commandId	The ID of the command to listen for.
callback	A function to execute when the command is invoked.

## Details

RStudio commands can be invoked from menus, toolbars, keyboard shortcuts, and the Command Palette, as well as the RStudio API. The callback will be executed whenever the command is invoked, regardless of how the invocation was triggered.

See the RStudio Server Professional Administration Guide appendix for a list of supported command IDs.

The callback is executed *after* the command has been run, but as some commands initiate asynchronous processes, there is no guarantee that the command has finished its work when the callback is invoked.

If you're having trouble figuring out the ID of a command you want to listen for, it can be helpful to discover it by listening to the full command stream; see the example in [registerCommandStreamCallback](#) for details.

Note that no error will be raised if you use a command ID that does not exist.

## Value

A handle representing the registration. Pass this handle to [unregisterCommandCallback](#) to unregister the callback.

**Note**

The registerCommandCallback function was introduced in RStudio 1.4.1589.

**See Also**

[unregisterCommandCallback](#) to unregister the callback, and [registerCommandStreamCallback](#) to be notified whenever *any* command is executed.

**Examples**

```
## Not run:
# Set up a callback to display an encouraging dialog whenever
# the user knits a document
handle <- rstudioapi::registerCommandCallback(
  "knitDocument",
  function() {
    rstudioapi::showDialog(
      "Achievement",
      "Congratulations, you have knitted a document. Well done."
    )
  })

# Knit the document interactively and observe the dialog

# Later: Unregister the callback
rstudioapi::unregisterCommandCallback(handle)

## End(Not run)
```

---

registerCommandStreamCallback

*Register Command Stream Callback*

---

**Description**

Registers a callback to be executed whenever any RStudio command is invoked.

**Usage**

```
registerCommandStreamCallback(callback)
```

**Arguments**

callback      A function to execute when the command is invoked.

### Details

The callback function will be given a single argument with the ID of the command that was invoked. See the RStudio Server Professional Administration Guide appendix for a list of command IDs.

Note that there is a small performance penalty incurred across the IDE when a command stream listener is present. If you only need to listen to a few specific commands, it is recommended to set up callbacks for them individually using [registerCommandCallback](#).

### Value

A handle representing the registration. Pass this handle to [unregisterCommandCallback](#) to unregister the callback.

### Note

The `registerCommandStreamCallback` function was introduced in RStudio 1.4.1589.

### See Also

[unregisterCommandCallback](#) to unregister the callback, and [registerCommandCallback](#) to be notified whenever a *specific* command is executed.

### Examples

```
## Not run:
# Set up a callback to print the ID of commands executed to the console.
handle <- rstudioapi::registerCommandStreamCallback(function(id) {
  message("Command executed: ", id)
})

# Later: Unregister the callback
rstudioapi::unregisterCommandCallback(handle)

## End(Not run)
```

---

removeTheme

*Remove a custom theme from RStudio.*

---

### Description

Remove a custom theme from RStudio.

### Usage

```
removeTheme(name)
```

### Arguments

name            The unique name of the theme to remove.

**Note**

The removeTheme function was introduced in RStudio 1.2.879.

---

restartSession	<i>Restart the R Session</i>
----------------	------------------------------

---

**Description**

Restart the RStudio session.

**Usage**

```
restartSession(command = "", clean = FALSE)
```

**Arguments**

command	A command (as a string) to be run after restarting.
clean	Boolean; when FALSE, the current R session (including loaded packages and data objects) will be saved and restored in the new session.

**Note**

The restartSession function was added in version 1.1.281 of RStudio. Support for the clean argument was added for version 2024.04 release of RStudio; it is silently ignored in older versions of RStudio.

---

rstudio-documents	<i>Interact with Documents open in RStudio</i>
-------------------	--

---

**Description**

Use these functions to interact with documents open in RStudio.

**Usage**

```
insertText(location = NULL, text = NULL, id = NULL)
modifyRange(location = NULL, text = NULL, id = NULL)
setDocumentContents(text, id = NULL)
setCursorPosition(position, id = NULL)
setSelectionRanges(ranges, id = NULL)
```

```

documentId(allowConsole = TRUE)

documentPath(id = NULL)

documentSave(id = NULL)

documentSaveAll()

documentNew(
  text,
  type = c("r", "rmarkdown", "sql"),
  position = document_position(0, 0),
  execute = FALSE
)

documentOpen(path, line = -1L, col = -1L, moveCursor = TRUE)

documentClose(id = NULL, save = TRUE)

```

### Arguments

location	An object specifying the positions, or ranges, wherein text should be inserted. See <b>Details</b> for more information.
text	A character vector, indicating what text should be inserted at each aforementioned range. This should either be length one (in which case, this text is applied to each range specified); otherwise, it should be the same length as the ranges list.
id	The document id. When NULL or blank, the requested operation will apply to the currently open, or last focused, RStudio document.
position	The cursor position, typically created through <code>document_position()</code> .
ranges	A list of one or more ranges, typically created through <code>document_range()</code> .
allowConsole	Allow the pseudo-id #console to be returned, if the R console is currently focused? Set this to FALSE if you'd always like to target the currently-active or last-active editor in the Source pane.
type	The type of document to be created.
execute	Should the code be executed after the document is created?
path	The path to the document.
line	The line in the document to navigate to.
col	The column in the document to navigate to.
moveCursor	Boolean; move the cursor to the requested location after opening the document?
save	Whether to commit unsaved changes to the document before closing it.

### Details

location should be a (list of) `document_position` or `document_range` object(s), or numeric vectors coercable to such objects.

To operate on the current selection in a document, call `insertText()` with only a text argument, e.g.

```
insertText("# Hello\n")
insertText(text = "# Hello\n")
```

Otherwise, specify a (list of) positions or ranges, as in:

```
# insert text at the start of the document
insertText(c(1, 1), "# Hello\n")

# insert text at the end of the document
insertText(Inf, "# Hello\n")

# comment out the first 5 rows
pos <- Map(c, 1:5, 1)
insertText(pos, "# ")

# uncomment the first 5 rows, undoing the previous action
rng <- Map(c, Map(c, 1:5, 1), Map(c, 1:5, 3))
modifyRange(rng, "")
```

`modifyRange` is a synonym for `insertText`, but makes its intent clearer when working with ranges, as performing text insertion with a range will replace the text previously existing in that range with new text. For clarity, prefer using `insertText` when working with [document\\_positions](#), and `modifyRange` when working with [document\\_ranges](#).

`documentClose` accepts an ID of an open document rather than a path. You can retrieve the ID of the active document using the `documentId()` function.

Closing is always done non-interactively; that is, no prompts are given to the user. If the user has made changes to the document but not saved them, then the `save` parameter governs the behavior: when `TRUE`, unsaved changes are committed, and when `FALSE` they are discarded.

## Note

The `insertText`, `modifyRange` and `setDocumentContents` functions were added with version 0.99.796 of RStudio.

The `setCursorPosition` and `setSelectionRanges` functions were added with version 0.99.1111 of RStudio.

The `documentSave` and `documentSaveAll` functions were added with version 1.1.287 of RStudio.

The `documentId` and `documentPath` functions were added with version 1.4.843 of RStudio.

The `documentNew` function was introduced in RStudio 1.2.640.

The `documentOpen` function was introduced in RStudio 1.4.1106.

The `documentClose` function was introduced in RStudio 1.2.1255

---

`rstudio-editors`*Retrieve Information about an RStudio Editor*

---

## Description

Returns information about an RStudio editor.

## Usage

```
getActiveDocumentContext()
```

```
getSourceEditorContext(id = NULL)
```

```
getConsoleEditorContext()
```

## Arguments

<code>id</code>	The ID of a particular document, as retrieved by <code>documentId()</code> or similar. Supported in RStudio 2022.06.0 or newer.
-----------------	---

## Details

The selection field returned is a list of document selection objects. A document selection is just a pairing of a document range, and the text within that range.

## Value

A list with elements:

<code>id</code>	The document ID.
<code>path</code>	The path to the document on disk.
<code>contents</code>	The contents of the document.
<code>selection</code>	A list of selections. See <b>Details</b> for more information.

## Note

The `getActiveDocumentContext` function was added with version 0.99.796 of RStudio, while the `getSourceEditorContext` and the `getConsoleEditorContext` functions were added with version 0.99.1111.



---

savePlotAsImage	<i>Save active RStudio plot image</i>
-----------------	---------------------------------------

---

**Description**

Save the plot currently displayed in the Plots pane as an image.

**Usage**

```
savePlotAsImage(
  file,
  format = c("png", "jpeg", "bmp", "tiff", "emf", "svg", "eps"),
  width,
  height
)
```

**Arguments**

file	The target file path.
format	The Image format. Must be one of ("png", "jpeg", "bmp", "tiff", "emf", "svg", or "eps").
width	The image width, in pixels.
height	The image height, in pixels.

**Note**

The savePlotAsImage function was introduced in RStudio 1.1.57.

---

selections	<i>Manipulate User Selections in the RStudio IDE</i>
------------	--

---

**Description**

These functions allow users of the `rstudioapi` package to read and write the user's current selection within the RStudio IDE.

**Usage**

```
selectionGet(id = NULL)

selectionSet(value = NULL, id = NULL)
```

**Arguments**

id	The document ID. When NULL (the default), the active, or most-recently edited, document will be used.
value	The text contents to set for the selection.

---

sendToConsole	<i>Send code to the R console</i>
---------------	-----------------------------------

---

**Description**

Send code to the R console, and optionally execute it.

**Usage**

```
sendToConsole(code, execute = TRUE, echo = TRUE, focus = TRUE, animate = FALSE)
```

**Arguments**

code	The R code to be executed, as a character vector.
execute	Boolean; should the code be executed after being submitted to the console? If FALSE, code is submitted to the console but is not executed.
echo	Boolean; echo the code in the console as it is executed?
focus	Boolean; focus the console after sending code?
animate	Boolean; should the submitted code be animated, as if someone was typing it?

**Note**

The sendToConsole function was added in version 0.99.787 of RStudio.

**Examples**

```
## Not run:  
rstudioapi::sendToConsole(".Platform", execute = FALSE, animate = TRUE)  
  
## End(Not run)
```

---

setGhostText	<i>Set ghost text</i>
--------------	-----------------------

---

**Description**

Set ghost text in the current document. The ghost text will be inserted at the current cursor position. Ghost text can be inserted into the document by pressing Tab, and will be automatically dismissed if the user navigates the cursor away.

**Usage**

```
setGhostText(text)
```

**Arguments**

text            The ghost text to set.

---

showDialog            *Show Dialog Box*

---

**Description**

Shows a dialog box with a given title and contents.

**Usage**

```
showDialog(title, message, url = "")
```

**Arguments**

title            The title to display in the dialog box.  
message            A character vector with the contents to display in the main dialog area. Contents can contain the following HTML tags: "p", "em", "strong", "b" and "i".  
url                An optional url to display under the message.

**Details**

```
showDialog("A dialog", "Showing <b>bold</b> text in the  
message.")
```

**Note**

The showDialog function was added in version 1.1.67 of RStudio.

---

showPrompt            *Show Prompt Dialog Box*

---

**Description**

Shows a dialog box with a prompt field.

**Usage**

```
showPrompt(title, message, default = NULL)
```

**Arguments**

title            The title to display in the dialog box.  
message            A character vector with the contents to display in the main dialog area.  
default            An optional character vector that fills the prompt field with a default value.

**Note**

The showPrompt function was added in version 1.1.67 of RStudio.

---

showQuestion	<i>Show Question Dialog Box</i>
--------------	---------------------------------

---

**Description**

Shows a dialog box asking a question.

**Usage**

```
showQuestion(title, message, ok = NULL, cancel = NULL)
```

**Arguments**

title	The title to display in the dialog box.
message	A character vector with the contents to display in the main dialog area.
ok	And optional character vector that overrides the caption for the OK button.
cancel	An optional character vector that overrides the caption for the Cancel button.

**Note**

The showQuestion function was added in version 1.1.67 of RStudio.

---

sourceMarkers	<i>Display source markers</i>
---------------	-------------------------------

---

**Description**

Display user navigable source markers in a pane within RStudio.

**Usage**

```
sourceMarkers(  
  name,  
  markers,  
  basePath = NULL,  
  autoSelect = c("none", "first", "error")  
)
```

**Arguments**

name	The name of marker set. If there is a market set with this name already being shown, those markers will be replaced.
markers	An R list, or data.frame, of source markers. See <b>details</b> for more details on the expected format.
basePath	Optional. If all source files are within a base path, then specifying that path here will result in file names being displayed as relative paths. Note that in this case markers still need to specify source file names as full paths.
autoSelect	Auto-select a marker after displaying the marker set?

**Details**

The markers argument can contains either a list of marker lists or a data frame with the appropriate marker columns. The fields in a marker are as follows (all are required):

type	The marker type ("error", "warning", "info", "style", or "usage").
file	The path to the associated source file.
line	The line number for the associated marker.
column	The column number for the associated marker.
message	A message associated with the marker at this location.

Note the marker message can contain ANSI SGR codes for formatting. The cli package can format text for style and color.

**Note**

The sourceMarkers function was added in version 0.99.225 of RStudio.

---

systemUsername	<i>Get System Username</i>
----------------	----------------------------

---

**Description**

Returns the system username of the current user.

**Usage**

```
systemUsername()
```

---

terminalActivate	<i>Activate Terminal</i>
------------------	--------------------------

---

### Description

Ensure terminal is running and optionally bring to front in RStudio.

### Usage

```
terminalActivate(id = NULL, show = TRUE)
```

### Arguments

id	The terminal id. The id is obtained from <a href="#">terminalList()</a> , <a href="#">terminalVisible()</a> , <a href="#">terminalCreate()</a> , or <a href="#">terminalExecute()</a> . If NULL, the terminal tab will be selected but no specific terminal will be chosen.
show	If TRUE, bring the terminal to front in RStudio.

### Note

The terminalActivate function was added in version 1.1.350 of RStudio.

### Examples

```
## Not run:
# create a hidden terminal and run a lengthy command
termId = rstudioapi::terminalCreate(show = FALSE)
rstudioapi::terminalSend(termId, "sleep 5\n")

# wait until a busy terminal is finished
while (rstudioapi::terminalBusy(termId)) {
  Sys.sleep(0.1)
}
print("Terminal available")#'

rstudioapi::terminalActivate(termId)

## End(Not run)
```

---

terminalBuffer	<i>Get Terminal Buffer</i>
----------------	----------------------------

---

**Description**

Returns contents of a terminal buffer.

**Usage**

```
terminalBuffer(id, stripAnsi = TRUE)
```

**Arguments**

id	The terminal id. The id is obtained from <a href="#">terminalList()</a> , <a href="#">terminalVisible()</a> , <a href="#">terminalCreate()</a> , or <a href="#">terminalExecute()</a> .
stripAnsi	If FALSE, don't strip out Ansi escape sequences before returning terminal buffer.

**Value**

The terminal contents, one line per row.

**Note**

The terminalBuffer function was added in version 1.1.350 of RStudio.

---

terminalBusy	<i>Is Terminal Busy</i>
--------------	-------------------------

---

**Description**

Are terminals reporting that they are busy?

**Usage**

```
terminalBusy(id)
```

**Arguments**

id	The terminal id. The id is obtained from <a href="#">terminalList()</a> , <a href="#">terminalVisible()</a> , <a href="#">terminalCreate()</a> , or <a href="#">terminalExecute()</a> .
----	---

**Details**

This feature is only supported on RStudio Desktop for Mac and Linux, and RStudio Server. It always returns FALSE on RStudio Desktop for Microsoft Windows.

**Value**

a boolean

**Note**

The terminalBusy function was added in version 1.1.350 of RStudio.

**Examples**

```
## Not run:
# create a hidden terminal and run a lengthy command
termId <- rstudioapi::terminalCreate(show = FALSE)
rstudioapi::terminalSend(termId, "sleep 5\n")

# wait until a busy terminal is finished
while (rstudioapi::terminalBusy(termId)) {
  Sys.sleep(0.1)
}
print("Terminal available")

## End(Not run)
```

---

terminalClear

*Clear Terminal Buffer*

---

**Description**

Clears the buffer for specified terminal.

**Usage**

```
terminalClear(id)
```

**Arguments**

id                   The terminal id. The id is obtained from [terminalList\(\)](#), [terminalVisible\(\)](#), [terminalCreate\(\)](#), or [terminalExecute\(\)](#).

**Note**

The terminalClear function was added in version 1.1.350 of RStudio.



**Examples**

```
## Not run:
termId <- rstudioapi::terminalCreate()
rstudioapi::terminalSend(termId, 'ls -l\n')
Sys.sleep(3)
rstudioapi::terminalClear(termId)

## End(Not run)
```

---

terminalContext	<i>Retrieve Information about RStudio Terminals</i>
-----------------	---

---

**Description**

Returns information about RStudio terminal instances.

**Usage**

```
terminalContext(id)
```

**Arguments**

`id` The terminal id. The id is obtained from [terminalList\(\)](#), [terminalVisible\(\)](#), [terminalCreate\(\)](#), or [terminalExecute\(\)](#).

**Value**

A list with elements:

handle	the internal handle
caption	caption
title	title set by the shell
working_dir	working directory
shell	shell type
running	is terminal process executing
busy	is terminal running a program
exit_code	process exit code or NULL
connection	websockets or rpc
sequence	creation sequence
lines	lines of text in terminal buffer
cols	columns in terminal
rows	rows in terminal
pid	process id of terminal shell
full_screen	full screen program running

**Note**

The terminalContext function was added in version 1.1.350 of RStudio.

**Examples**

```
## Not run:  
termId <- rstudioapi::terminalCreate("example", show = FALSE)  
View(rstudioapi::terminalContext(termId))
```

```
## End(Not run)
```

---

terminalCreate	<i>Create a Terminal</i>
----------------	--------------------------

---

**Description**

Create a new Terminal.

**Usage**

```
terminalCreate(caption = NULL, show = TRUE, shellType = NULL)
```

**Arguments**

caption	The desired terminal caption. When NULL or blank, the terminal caption will be chosen by the system.
show	If FALSE, terminal won't be brought to front.
shellType	Shell type for the terminal: NULL or "default" to use the shell selected in Global Options. For Microsoft Windows, alternatives are "win-cmd" for 64-bit Command Prompt, "win-ps" for 64-bit PowerShell, "win-git-bash" for Git Bash, or "win-wsl-bash" for Bash on Windows Subsystem for Linux. On Linux, Mac, and RStudio Server "custom" will use the custom terminal defined in Global Options. If the requested shell type is not available, the default shell will be used, instead.

**Value**

The terminal identifier as a character vector (NULL if unable to create the terminal or the given terminal caption is already in use).

**Note**

The terminalCreate function was added in version 1.1.350 of RStudio and the ability to specify shellType was added in version 1.2.696.

**Examples**

```
## Not run:
termId <- rstudioapi::terminalCreate('My Terminal')

## End(Not run)
```

---

terminalExecute	<i>Execute Command</i>
-----------------	------------------------

---

**Description**

Execute a command, showing results in the terminal pane.

**Usage**

```
terminalExecute(command, workingDir = NULL, env = character(), show = TRUE)
```

**Arguments**

command	System command to be invoked, as a character string.
workingDir	Working directory for command
env	Vector of name=value strings to set environment variables
show	If FALSE, terminal won't be brought to front

**Value**

The terminal identifier as a character vector (NULL if unable to create the terminal).

**Note**

The terminalExecute function was added in version 1.1.350 of RStudio.

**Examples**

```
## Not run:
termId <- rstudioapi::terminalExecute(
  command = 'echo $HELLO && echo $WORLD',
  workingDir = '/usr/local',
  env = c('HELLO=WORLD', 'WORLD=EARTH'),
  show = FALSE)

while (is.null(rstudioapi::terminalExitCode(termId))) {
  Sys.sleep(0.1)
}

result <- terminalBuffer(termId)
```

```
terminalKill(termId)
print(result)

## End(Not run)
```

---

terminalExitCode	<i>Terminal Exit Code</i>
------------------	---------------------------

---

**Description**

Get exit code of terminal process, or NULL if still running.

**Usage**

```
terminalExitCode(id)
```

**Arguments**

`id` The terminal id. The id is obtained from [terminalList\(\)](#), [terminalVisible\(\)](#), [terminalCreate\(\)](#), or [terminalExecute\(\)](#).

**Value**

The exit code as an integer vector, or NULL if process still running.

**Note**

The terminalExitCode function was added in version 1.1.350 of RStudio.

---

terminalKill	<i>Kill Terminal</i>
--------------	----------------------

---

**Description**

Kill processes and close a terminal.

**Usage**

```
terminalKill(id)
```

**Arguments**

`id` The terminal id. The id is obtained from [terminalList\(\)](#), [terminalVisible\(\)](#), [terminalCreate\(\)](#), or [terminalExecute\(\)](#).

**Note**

The terminalKill function was added in version 1.1.350 of RStudio.

---

terminalList	<i>Get All Terminal Ids</i>
--------------	-----------------------------

---

**Description**

Return a character vector containing all the current terminal identifiers.

**Usage**

```
terminalList()
```

**Value**

The terminal identifiers as a character vector.

**Note**

The terminalList function was added in version 1.1.350 of RStudio.

---

terminalRunning	<i>Is Terminal Running</i>
-----------------	----------------------------

---

**Description**

Does a terminal have a process associated with it? If the R session is restarted after a terminal has been created, the terminal will not restart its shell until it is displayed either via the user interface, or via `terminalActivate()`.

**Usage**

```
terminalRunning(id)
```

**Arguments**

id	The terminal id. The id is obtained from <code>terminalList()</code> , <code>terminalVisible()</code> , <code>terminalCreate()</code> , or <code>terminalExecute()</code> .
----	---

**Value**

a boolean

**Note**

The terminalRunning function was added in version 1.1.350 of RStudio.

## Examples

```
## Not run:
# termId has a handle to a previously created terminal
# make sure it is still running before we send it a command
if (!rstudioapi::terminalRunning(termId)) {
  rstudioapi::terminalActivate(termId)

  # wait for it to start
  while (!rstudioapi::terminalRunning(termId)) {
    Sys.sleep(0.1)
  }

  terminalSend(termId, "echo Hello\n")
}

## End(Not run)
```

---

terminalSend	<i>Send Text to a Terminal</i>
--------------	--------------------------------

---

## Description

Send text to an existing terminal.

## Usage

```
terminalSend(id, text)
```

## Arguments

id	The terminal id. The id is obtained from <a href="#">terminalList()</a> , <a href="#">terminalVisible()</a> , <a href="#">terminalCreate()</a> , or <a href="#">terminalExecute()</a> .
text	Character vector containing text to be inserted.

## Note

The terminalSend function was added in version 1.1.350 of RStudio.

## Examples

```
## Not run:
termId <- rstudioapi::terminalCreate()
rstudioapi::terminalSend(termId, 'ls -l\n')

## End(Not run)
```

---

terminalVisible	<i>Get Visible Terminal</i>
-----------------	-----------------------------

---

**Description**

Get Visible Terminal

**Usage**

```
terminalVisible()
```

**Value**

Terminal identifier selected in the client, if any.

**Note**

The terminalVisible function was added in version 1.1.350 of RStudio.

---

translateLocalUrl	<i>Translate Local URL</i>
-------------------	----------------------------

---

**Description**

Translates a local URL into an externally accessible URL on RStudio Server.

**Usage**

```
translateLocalUrl(url, absolute = FALSE)
```

**Arguments**

url	The fully qualified URL to translate; for example, <code>http://localhost:1234/service/page.html</code> .
absolute	Whether to return a relative path URL (the default) or an absolute URL.

**Details**

On RStudio Server, URLs which refer to the local host network address (such as `http://localhost:1234/` and `http://127.0.0.1:5678/`) must be translated in order to be externally accessible from a browser. This method performs the required translation, and returns the translated URL, which RStudio Server uses to proxy HTTP requests.

Returns an unmodified URL on RStudio Desktop, and when the URL does not refer to a local address.

**Value**

The translated URL.

---

unregisterCommandCallback  
*Unregister Command Callback*

---

**Description**

Removes a previously registered command callback.

**Usage**

```
unregisterCommandCallback(handle)
```

**Arguments**

handle            The registration handle to remove.

**Value**

NULL, invisibly.

**Note**

The unregisterCommandCallback function was introduced in RStudio 1.4.1589.

---

updateDialog            *Updates a Dialog Box*

---

**Description**

Updates specific properties from the current dialog box.

**Usage**

```
updateDialog(...)
```

**Arguments**

...                Named parameters and values to update a dialog box.

**Details**

Currently, the only dialog with support for this action is the New Connection dialog in which the code preview can be updated through this API.

```
updateDialog(code = "con <- NULL")
```

**Note**

The updateDialog function was added in version 1.1.67 of RStudio.



---

userIdentity	<i>Get User Identity</i>
--------------	--------------------------

---

**Description**

Returns the identity (displayed name) of the current user.

**Usage**

```
userIdentity()
```

---

versionInfo	<i>RStudio version information</i>
-------------	------------------------------------

---

**Description**

Query information about the currently running instance of RStudio.

**Usage**

```
versionInfo()
```

**Value**

An R list with the following elements:

version	The version of RStudio.
mode	"desktop" for RStudio Desktop, or "server" for RStudio Server.
citation	Information on how RStudio can be cited in academic publications.

**Note**

The versionInfo function was added in version 0.97.124 of RStudio.

**Examples**

```
## Not run:
info <- rstudioapi::versionInfo()

# check what version of RStudio is in use
if (info$version >= "1.4") {
  # code specific to versions of RStudio 1.4 and newer
}

# check whether RStudio Desktop or RStudio Server is being used
if (info$mode == "desktop") {
```

```

    # code specific to RStudio Desktop
  }

  # Get the citation
  info$citation

## End(Not run)

```

---

viewer

*View local web content within RStudio*


---

### Description

View local web content within RStudio. Content can be served from static files in the R session temporary directory, or via a web application running on localhost.

### Usage

```
viewer(url, height = NULL)
```

### Arguments

url	Application URL. This can be either a localhost URL or a path to a file within the R session temporary directory (i.e. a path returned by <code>tempfile()</code> ).
height	Desired height. Specifies a desired height for the Viewer pane (the default is NULL which makes no change to the height of the pane). This value can be numeric or the string "maximize" in which case the Viewer will expand to fill all vertical space. See details below for a discussion of constraints imposed on the height.

### Details

RStudio also sets the global viewer option to the `rstudioapi::viewer` function so that it can be invoked in a front-end independent manner.

Applications are displayed within the Viewer pane. The application URL must either be served from localhost or be a path to a file within the R session temporary directory. If the URL doesn't conform to these requirements it is displayed within a standard browser window.

The height parameter specifies a desired height, however it's possible the Viewer pane will end up smaller if the request can't be fulfilled (RStudio ensures that the pane paired with the Viewer maintains a minimum height). A height of 400 pixels or lower is likely to succeed in a large proportion of configurations.

A very large height (e.g. 2000 pixels) will allocate the maximum allowable space for the Viewer (while still preserving some view of the pane above or below it). The value "maximize" will force the Viewer to full height. Note that this value should only be specified in cases where maximum vertical space is essential, as it will result in one of the user's other panes being hidden.

## Viewer Detection

When a page is displayed within the Viewer it's possible that the user will choose to pop it out into a standalone browser window. When rendering inside a standard browser you may want to make different choices about how content is laid out or scaled. Web pages can detect that they are running inside the Viewer pane by looking for the `viewer_pane` query parameter, which is automatically injected into URLs when they are shown in the Viewer. For example, the following URL:

```
http://localhost:8100
```

When rendered in the Viewer pane is transformed to:

```
http://localhost:8100?viewer_pane=1
```

To provide a good user experience it's strongly recommended that callers take advantage of this to automatically scale their content to the current size of the Viewer pane. For example, re-rendering a JavaScript plot with new dimensions when the size of the pane changes.

## Note

The `viewer` function was added in version 0.98.423 of RStudio. The ability to specify `maximize` for the `height` parameter was introduced in version 0.99.1001 of RStudio.

## Examples

```
## Not run:

# run an application inside the IDE
rstudioapi::viewer("http://localhost:8100")

# run an application and request a height of 500 pixels
rstudioapi::viewer("http://localhost:8100", height = 500)

# use 'viewer' option if set, or `utils::browseURL()` if unset
viewer <- getOption("viewer", default = utils::browseURL)
viewer("http://localhost:8100")

# generate a temporary html file and display it
dir <- tempfile()
dir.create(dir)
htmlFile <- file.path(dir, "index.html")
# (code to write some content to the file)
rstudioapi::viewer(htmlFile)

## End(Not run)
```

writePreference      *Write Preference*

---

**Description**

Writes a user preference, useful to remember preferences across different R sessions for the same user.

**Usage**

```
writePreference(name, value)
```

**Arguments**

name	The name of the preference.
value	The value of the preference.

**Note**

The writePreference function was added in version 1.1.67 of RStudio.

**See Also**

[writeRStudioPreference](#), which changes RStudio IDE preferences.

---

writeRStudioPreference  
*Write RStudio Preference*

---

**Description**

Writes an internal RStudio IDE preference for the current user.

**Usage**

```
writeRStudioPreference(name, value)
```

**Arguments**

name	The name of the preference.
value	The value of the preference.

**Details**

RStudio IDE internal preferences include the values displayed in RStudio's Global Options dialog as well as a number of additional settings. Set them carefully; inappropriate values can cause unexpected behavior. See the RStudio Server Professional Administration Guide appendix for your version of RStudio for a full list of preference names and values.

**Note**

The `writeRStudioPreference` function was added in version 1.3.387 of RStudio.

**See Also**

[writePreference](#), which can be used to store arbitrary user (non-RStudio) preferences.  
[readRStudioPreference](#), which reads internal RStudio IDE preferences.

**Examples**

```
## Not run:  
# Hide RStudio's toolbar.  
rstudioapi::writeRStudioPreference("toolbar_visible", FALSE)  
  
## End(Not run)
```

# Index

## \* **job-launcher functionality**

- launcherAvailable, 28
- launcherConfig, 28
- launcherContainer, 29
- launcherControlJob, 29
- launcherGetInfo, 30
- launcherGetJob, 30
- launcherGetJobs, 31
- launcherHostMount, 31
- launcherNfsMount, 32
- launcherPlacementConstraint, 33
- launcherResourceLimit, 33
- launcherSubmitJob, 34
- launcherSubmitR, 36

## \* **jobs**

- jobAdd, 21
- jobAddOutput, 23
- jobAddProgress, 23
- jobGetState, 24
- jobList, 24
- jobRemove, 25
- jobRunScript, 25
- jobSetProgress, 26
- jobSetState, 26
- jobSetStatus, 27

addTheme, 3

applyTheme, 4

as.document\_position  
(document\_position), 11

as.document\_range (document\_range), 11

askForPassword, 5

askForSecret, 5

bugReport, 6

build-tools, 6

buildToolsCheck (build-tools), 6

buildToolsExec (build-tools), 6

buildToolsInstall (build-tools), 6

callFun, 7

chunk-callbacks, 8

convertTheme, 8

createProjectTemplate, 9

dictionaries, 10

dictionariesPath (dictionaries), 10

document\_position, 11, 11, 12, 46, 47

document\_range, 11, 46, 47

documentClose (rstudio-documents), 45

documentId (rstudio-documents), 45

documentNew (rstudio-documents), 45

documentOpen (rstudio-documents), 45

documentPath (rstudio-documents), 45

documentSave (rstudio-documents), 45

documentSaveAll (rstudio-documents), 45

documentSaveAll(), 40

executeCommand, 12

exists, 18

file-dialogs, 13

filesPaneNavigate, 14

findFun (hasFun), 18

get, 18

getActiveDocumentContext  
(rstudio-editors), 48

getActiveProject, 14

getConsoleEditorContext  
(rstudio-editors), 48

getDelegatedAzureToken, 15

getPersistentValue (persistent-values),  
37

getRStudioPackageDependencies, 15

getSourceEditorContext  
(rstudio-editors), 48

getThemeInfo, 16

getThemes, 16

getVersion, 17

- hasColorConsole, 17
- hasFun, 7, 18
- highlightUi, 19
  
- initializeProject (projects), 40
- insertText (rstudio-documents), 45
- is.document\_position
  - (document\_position), 11
- is.document\_range (document\_range), 11
- isAvailable, 20
- isBackgroundJob (isJob), 21
- isJob, 21
- isWorkbenchJob (isJob), 21
  
- jobAdd, 21, 23–27
- jobAddOutput, 22, 23, 23, 24–27
- jobAddProgress, 22, 23, 23, 24–27
- jobGetState, 22–24, 24, 25–27
- jobList, 22–24, 24, 25–27
- jobRemove, 22–24, 25, 26, 27
- jobRunScript, 22–25, 25, 26, 27
- jobSetProgress, 22–26, 26, 27
- jobSetState, 22–26, 26, 27
- jobSetStatus, 22–27, 27
  
- launcherAvailable, 28, 28, 29–36
- launcherConfig, 28, 28, 29–36
- launcherConfig(), 35
- launcherContainer, 28, 29, 30–36
- launcherControlJob, 28, 29, 29, 30–36
- launcherGetInfo, 28–30, 30, 31–36
- launcherGetJob, 28–30, 30, 31–36
- launcherGetJobs, 28–30, 31, 32–36
- launcherHostMount, 28–31, 31, 32–36
- launcherHostMount(), 35
- launcherNfsMount, 28–32, 32, 33–36
- launcherNfsMount(), 35
- launcherPlacementConstraint, 28–32, 33, 34–36
- launcherPlacementConstraint(), 35
- launcherResourceLimit, 28–33, 33, 35, 36
- launcherResourceLimit(), 35
- launcherSubmitJob, 28–34, 34, 36
- launcherSubmitJob(), 28, 29, 31–33, 36
- launcherSubmitR, 28–35, 36
  
- modifyRange (rstudio-documents), 45
  
- navigateToFile, 36
  
- numeric\_version, 17
  
- openProject (projects), 40
  
- persistent-values, 37
- previewRd, 38
- previewSql, 39
- primary\_selection, 39
- projects, 40
  
- readPreference, 40, 41
- readRStudioPreference, 41, 41, 69
- registerChunkCallback
  - (chunk-callbacks), 8
- registerCommandCallback, 13, 42, 44
- registerCommandStreamCallback, 42, 43, 43
- removeTheme, 44
- restartSession, 45
- rstudio-documents, 45
- rstudio-editors, 48
  
- savePlotAsImage, 49
- selectDirectory (file-dialogs), 13
- selectFile (file-dialogs), 13
- selectionGet (selections), 49
- selections, 49
- selectionSet (selections), 49
- sendToConsole, 50
- setCursorPosition (rstudio-documents), 45
- setDocumentContents
  - (rstudio-documents), 45
- setGhostText, 50
- setPersistentValue (persistent-values), 37
- setSelectionRanges (rstudio-documents), 45
- showDialog, 51
- showPrompt, 51
- showQuestion, 52
- sourceMarkers, 52
- systemUsername, 53
  
- tempfile(), 66
- terminalActivate, 54, 61
- terminalBuffer, 55
- terminalBusy, 55
- terminalClear, 56

- terminalContext, [57](#)
- terminalCreate, [54–57](#), [58](#), [60–62](#)
- terminalExecute, [54–57](#), [59](#), [60–62](#)
- terminalExitCode, [60](#)
- terminalKill, [60](#)
- terminalList, [54–57](#), [60](#), [61](#), [61](#), [62](#)
- terminalRunning, [61](#)
- terminalSend, [62](#)
- terminalVisible, [54–57](#), [60–62](#), [63](#)
- translateLocalUrl, [63](#)
  
- unregisterChunkCallback
  - (chunk-callbacks), [8](#)
- unregisterCommandCallback, [42–44](#), [64](#)
- updateDialog, [64](#)
- userDictionariesPath (dictionaries), [10](#)
- userIdentity, [65](#)
  
- verifyAvailable (isAvailable), [20](#)
- versionInfo, [65](#)
- View, [37](#)
- viewer, [66](#)
  
- writePreference, [41](#), [68](#), [69](#)
- writeRStudioPreference, [68](#), [68](#)