Package 'rstudioapi'

September 7, 2017

Title Safely Access the RStudio API

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

Version 0.7

Maintainer JJ Allaire <jj@rstudio.com>

License MIT + file LICENSE

URL https://github.com/rstudio/rstudioapi

BugReports https://github.com/rstudio/rstudioapi/issues

RoxygenNote 6.0.1

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author JJ Allaire [aut, cre],
Hadley Wickham [aut],
Kevin Ushey [aut],
Gary Ritchie [aut],
RStudio [cph]

Repository CRAN

Date/Publication 2017-09-07 21:41:46 UTC

R topics documented:

`askForPassword` *Ask the user for a password interactively*

Description

Ask the user for a password interactively.

Usage

```
askForPassword(prompt)
```

Arguments

prompt	Single element character vector containing the prompt to be displayed

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)
```

---

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")
}
```

createProjectTemplate *Create a Project Template*

Description

Create a project template. See 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 R package sources.
binding	The R 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 **New Project...** wizard.
subtitle	(optional) The subtitle to be shown within the **New Project...** 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 editting after creation?

document_position *Create a Document Position*

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 *Create a Range*

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 document_position indicating the start of the range.
end	A document_position indicating the end of the range.
x	An object coercable to document_range.

Value

An R list with class document_range and fields:

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

`file-dialogs` *Select a File / Folder*

Description

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

Usage

```
selectFile(caption = "Select File", label = "Select", path = NULL,
  filter = NULL, existing = TRUE)

selectDirectory(caption = "Select Directory", label = "Select",
  path = NULL)
```

Arguments

`caption`	The window title.
`label`	The label to use for the 'Accept' / 'OK' button.
`path`	The initial working directory, from which the file dialog should begin browsing. When `NULL`, defaults to the current RStudio project directory.
`filter`	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 `R Files (*.R)` here.
`existing`	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.

getActiveProject *Path to Active RStudio Project*

Description

Returns the path to the currently active RStudio project.

Usage

```
getActiveProject()
```

Value

Returns a single element character vector with the path of the currently active RStudio project. Returns `NULL` if no project is active.

Note

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

Examples

```
## Not run:
rstudioapi::getActiveProject()

## End(Not run)
```

getThemeInfo *Retrieve Themes*

Description

Retrieves a list with themes information. Currently, `editor` as the theme used under the code editor, `global` as the global theme applied to the main user interface in RStudio and `dark` when the user interface is optimized for dark themes.

Usage

```
getThemeInfo()
```

getVersion *Return the current version of the RStudio API*

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 *Check if Console Supports ANSI Color Escapes*

Description

Check if Console Supports ANSI Color Escapes

Usage

```
hasColorConsole()
```

Value

a boolean

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 *Exists/get for RStudio functions*

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

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

Examples

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

---

isAvailable *Check if RStudio is running.*

Description

Check if RStudio is running.

Usage

```
isAvailable(version_needed = NULL)

verifyAvailable(version_needed = NULL)
```

Arguments

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

Value

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

Examples

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

---

navigateToFile *Navigate to File*

Description

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

Usage

```
navigateToFile(file, line = -1L, column = -1L)
```

Arguments

file	Path to the file to open)
line	Optional; integer specifying the line number on which to place the cursor
column	Optional; integer specifying the column number on which to place the cursor

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 line and column arguments are both equal to -1L (the default), then the cursor position in the document that is opened will be preserved.

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	Key name
value	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 *Preview an Rd topic in the Help pane*

Description

Preview an Rd topic in the Help pane

Usage

```
previewRd(rdFile)
```

Arguments

rdFile	Single element character vector containing the name of the Rd file to be displayed

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)
```

---

primary_selection *Extract the Primary Selection*

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 *Open a Project in RStudio*

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 *Read Preference*

Description

Reads a user interface 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.

Note

The readPreference function was added in version 1.1.67 of RStudio.

restartSession *Restart the R Session*

Description

Restart the RStudio R session.

Usage

```
restartSession(command = "")
```

Arguments

command	An R command (as a string) to be run after restarting R.

Note

The restartSession function was added in version 1.1.281 of RStudio.

rstudio-documents *Interact with Documents open in RStudio*

Description

Use these functions to interact with documents open in RStudio.

Usage

```
insertText(location, text, id = NULL)

modifyRange(location, text, id = NULL)

setDocumentContents(text, id = NULL)

setCursorPosition(position, id = NULL)

setSelectionRanges(ranges, id = NULL)

documentSave(id = NULL)

documentSaveAll()
```

Arguments

location	An object specifying the positions, or ranges, wherein text should be inserted. See **Details** 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 mutation will apply to the currently open, or last focused, RStudio document. Use the id returned from getActiveDocumentContext() to ensure that the operation is applied on the intended document.
position	The cursor position, typically created through document_position().
ranges	A list of one or more ranges, typically created through document_range().

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.

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.

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

Description

Returns information about an RStudio editor.

Usage

```
getActiveDocumentContext()

getSourceEditorContext()

getConsoleEditorContext()
```

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:

`id`	The document ID.
`path`	The path to the document on disk.
`contents`	The contents of the document.
`selection`	A `list` of selections. See **Details** 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 *Save Active RStudio Plot as an Image*

Description

Save the currnently active RStudio as an image file.

Usage

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

Arguments

file	Target filename
format	Image format ("png", "jpeg", "bmp", "tiff", "emf", "svg", or "eps")
width	Image width in pixels
height	Image height in pixels

Note

The savePlotAsImage function was introduced in RStudio 1.1.57

sendToConsole *Send Code to the R Console*

Description

Send code to the R console and optionally execute it.

Usage

```
sendToConsole(code, execute = TRUE)
```

Arguments

code	Character vector containing code to be executed.
execute	TRUE to execute the code immediately.

Note

The sendToConsole function was added in version 0.99.787 of RStudio.

Examples

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

## End(Not run)
```

`showDialog` *Show Dialog Box*

Description

Shows a dialog box with a given title and contents.

Usage

```
showDialog(title, message, url = 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. Contents can contain the following HTML tags: "p", "em", "strong", "b" and "i".
`url`	And 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 *Show Question Dialog Box*

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 *Display Source Markers*

Description

Display user navigable source markers in a pane within RStudio

Usage

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

Arguments

name	Name of marker set (will replace any markers of the same name previously shown)
markers	List or data frame containing source markers (see below for details on how to specify markers)
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	Optional. Automatically select and drive focus to either the first marker or the first marker that is an error.

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	Marker type ("error", "warning", "info", "style", or "usage")
file	Path to source file
line	Line number witin source file
column	Column number within line
message	Short descriptive message

Note that if the message field is of class "html" (i.e. `inherits(message, "html") == TRUE`) then it's contents will be treated as HTML.

Note

The sourceMarkers function was added in version 0.99.225 of RStudio.

---

terminalActivate *Activate Terminal*

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 terminalList(), terminalVisible(), terminalCreate(), or terminalExecute(). 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` *Get Terminal Buffer*

Description

Returns contents of a terminal buffer.

Usage

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

Arguments

`id`	The terminal id. The `id` is obtained from `terminalList()`, `terminalVisible()`, `terminalCreate()`, or `terminalExecute()`.
`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` *Is Terminal Busy*

Description

Are terminals reporting that they are busy?

Usage

```
terminalBusy(id)
```

Arguments

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

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 *Retrieve Information about RStudio Terminals*

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` *Create a Terminal*

Description

Create a new Terminal.

Usage

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

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.

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.

Examples

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

## End(Not run)
```

---

terminalExecute *Execute Command*

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 *Terminal Exit Code*

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 *Kill Terminal*

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 *Get All Terminal Ids*

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` *Is Terminal Running*

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 `terminalList()`, `terminalVisible()`, `terminalCreate()`, or `terminalExecute()`.

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 *Send Text to a Terminal*

Description

Send text to an existing terminal.

Usage

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

Arguments

id	The terminal id. The id is obtained from terminalList(), terminalVisible(), terminalCreate(), or terminalExecute().
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 *Get Visible Terminal*

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.

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.

versionInfo *RStudio Version Information*

Description

Provides information about the currently running version of RStudio, including it's specific version number and whether it is running in desktop or server mode.

Usage

```
versionInfo()
```

Value

Returns a list with the following elements:

`version`	A package version object that can be used in comparisons. This is the same value which would be returned from
`mode`	Current program mode (either "desktop" or "server")
`citation`	An object inheriting from class `bibentry`

Note

The `versionInfo` function was added in version 0.97.124 of RStudio.

Examples

```
## Not run:
require(rstudioapi)
ver <- versionInfo()

# Test specific version constraint
if (ver$version >= "0.97") {
   # do some 0.97 dependent stuff
}

# Check current mode
desktopMode <- ver$mode == "desktop"
serverMode <- ver$mode == "server"

# Get the citation
ver$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 can be a Shiny, Rook, OpenCPU, or any other type of localhost web application.

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 `tempfile`).
`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)
```

```
# probe for viewer option then fall back to browseURL
viewer <- getOption("viewer")
if (!is.null(viewer))
   viewer("http://localhost:8100")
else
   utils::browseURL("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 interface 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.

Index