Package: svGUI (via r-universe)

October 8, 2026

Type Package

Version 1.0.2

Title 'SciViews::R' - Manage GUIs in R

Description Manage Graphical User Interfaces (GUI) in R. It is independent from any particular GUI widgets ('Tk', 'Gtk2', native, ...). It centralizes info about GUI elements currently used, and it dispatches GUI calls to the particular toolkits in use in function of the context (is R run at the terminal, within a 'Tk' application, a HTML page?).

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 2.6.0)

Suggests covr (>= 3.5.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

License GPL-2

URL https://github.com/SciViews/svGUI, https://www.sciviews.org/svGUI/, https://sciviews.r-universe.dev/svGUI

BugReports https://github.com/SciViews/svGUI/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

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

Date/Publication 2025-08-29 06:19:20 UTC

RemoteUrl https://github.com/SciViews/svGUI

RemoteRef HEAD

RemoteSha 54b97cae4ca7ffdc36de69da9e2313a0995e1214

Contents

svGUI-package 2
dont_ask 4
gui 5
gui_add 6
setUI 8

Index **10**

`svGUI-package` *'SciViews::R' - Manage GUIs in R*

Description

The 'SciViews' 'svGUI' package eases the management of Graphical User Interfaces (GUI) in R. It is independent from any particular GUI widgets ('Tk', 'Gtk2', native, ...). It centralizes info about GUI elements currently used, and it dispatches GUI calls to the particular toolkits in use in function of the context (is R run at the terminal, within a 'Tk' application, a HTML page?).

Details

The `gui` object defines a succession of GUI (or non-GUI) `widgets` to use. These could be `tcltk` (with the **'tcltk'** or **'tcltk2'** R packages), `gtk2` (with the **'RGtk2'** R package), `shiny`, etc. You are in charge of managing these different variants of your GUI. The `gui` object just defines the order of preference for those different variants, and to get a fallback mechanism in case your GUI is not implemented with given `widgets`. `.GUI` uses, by default, `widgets = c("nativeGUI", "textCLI")`. `"nativeGUI"` is, as you figure it out, a native version of the GUI element. A good example is `base::file.choose()` that displays a native dialog box to select a file. `"textCLI"` is **not** a GUI version, but a way to ask the same information to the user at the terminal (or Command Line, CLI). In the example, it could be `base::readline("File to use? ")`. You GUI should consider the proposed `widgets` in turn and use the first one on the list that is implemented. It is advised to implement also a version of `"textCLI"`, in case R is run in a text-only context. Finally, if none of the `widgets` can be run, your code should fall back to use a default value for the file, or to stop gracefully. This is required for non-interactive use or testing of your code, are when your GUI `ask` is set to `FALSE`.

Basic GUI items, like message boxes, input box, file or directory selectors, etc. could easily be implemented with different `widgets` and in a `"textCLI"` version (see the **'svDialogs'** package). So, if your GUI uses the present mechanisms, your end-user could choose the version of the dialog boxes he prefers to use, given the context (R run at the terminal, in 'RGui', in 'RStudio' or 'RStudio Server'; under Windows, Mac OS, or Linux, ...). The choice is easy: just change the sequence of `widgets` in the corresponding `gui` object. Of course, several `gui` objects can live together at the same time, providing different and independent contexts (say, one GUI build with **'RGtk2'** would favor `"gtk2"`, but another GUI using **'tcltk'** would either favor `"tcltk"` of course, or `c("nativeGUI", "tcltk")` just because native dialog boxes may look better, for instance, under macOS or Linux.

Finally, the `gui` object is basically a separate environment where you could also store various GUI-related objects. On one hand, it does not "pollute" other environments (the worse practice being to

put **'tcltk'**-related variables in the global environment), and on the other hand, it is very easy to get rid of all the GUI-related objects, just by `gui_remove("myGUI")`. Also, GUI-related items should not be `save.image()`d and `reload()`ed with the other objects, and the `gui`, being located outside of `.GlobalEnv` prevents it.

Methods

svGUI implements four methods for the S3 `gui` object:

`print()` Give information about the current state of the GUI

`$` Give access to various GUI properties or objects.

`startUI()` Start an UI action that requires to interrupt R (for instance, display an input dialog box) and manage to inform the `gui` object about it.

`setUI()` Change the `status` of the UI action currently running.

Important functions

`gui_add()` and `gui_change()` for construction and management of guis, `gui_remove()` to cleanly eliminate all GUI elements, `gui_list()` to list all `gui` objects currently loaded in the R session, `gui_widgets()` to manage the widgets this GUI can use, and in which order, `gui_ask()` allows to (temporary) disable UI actions to avoid any code that would require input from the user (e.g., to run in batch mode), `dont_ask()` to determine if the GUI cannot interrupt R to ask something to the user, and should proceed differently (say, just use a default value for an input).

Dispatch mechanism

Methods for `gui` objects can dispatch as usual using `amethod(..., gui = agui)` but note that these methods do not dispatch on the first provided argument, but to the named argument `gui`. There is another way to call `gui` methods: `agui$amethod(...)`. This may be a convenient alternative for those who prefer this style of calling object's methods (also used in reference classes, **'proto'** or **'R6'** objects).

Author(s)

Maintainer: Philippe Grosjean <phgrosjean@sciviews.org> (ORCID)

See Also

Useful links:

- https://github.com/SciViews/svGUI
- https://www.sciviews.org/svGUI/
- https://sciviews.r-universe.dev/svGUI
- Report bugs at https://github.com/SciViews/svGUI/issues

dont_ask *Can we interrupt R to ask something to the user though the GUI?*

Description

Determine if R code execution can be interrupted by the GUI, e.g., using a modal dialog box. It depends both on R being in `interactive()` mode and the `ask` flag of the GUI being set to `TRUE`.

Usage

```
dont_ask(gui = .GUI)

dontAsk(gui = .GUI)
```

Arguments

`gui`	A `gui` object.

Details

`dontAsk` and `dont_ask` are aliases.

Value

`TRUE` if the GUI cannot interrupt R. The function triggering the dialog box should then not be displayed and it should return the default value as the result. The function returns `TRUE` if R is run in a non interactive session, or if `ask` is set to `FALSE` for the GUI, or if it is not specified (`ask` is `NULL`) then `getOptions("gui.ask")` is used.

See Also

`gui_ask()`, `gui`

Examples

```
# What is the current state for the default GUI?
dont_ask()
```

gui *A GUI object.*

Description

The gui object contains and manages GUI-related data.

Usage

```
## S3 method for class 'gui'
gui$x

## S3 method for class 'gui'
print(x, ...)

is.gui(x)
```

Arguments

gui	A gui object..
x	An object or a function for $.
...	Further arguments (not used yet).

See Also

gui_add()

Examples

```
# Create a GUI
gui_add("myGUI")
is.gui(myGUI)
myGUI
# Put an object in the GUI environment (fake button)
myGUI$button <- "my_button"
# Retrieve it
myGUI$button
# Get the curent status of the GUI
myGUI$status
# Eliminate this GUI and all its objects
gui_remove("myGUI")
```

gui_add *Creation and management of GUI objects.*

Description

Create and manipulate gui objects to manage 'SciViews'-compatible GUIs (Graphical User Interfaces).

Usage

```
gui_add(gui.name = ".GUI", widgets = c("nativeGUI", "textCLI"), ask)

guiAdd(gui.name = ".GUI", widgets = c("nativeGUI", "textCLI"), ask)

gui_change(
  gui.name = ".GUI",
  widgets = c("nativeGUI", "textCLI"),
  reset = FALSE,
  ask
)

guiChange(
  gui.name = ".GUI",
  widgets = c("nativeGUI", "textCLI"),
  reset = FALSE,
  ask
)

gui_remove(gui.name)

guiRemove(gui.name)

gui_list()

guiList()

gui_widgets(gui, gui.name = ".GUI")

guiWidgets(gui, gui.name = ".GUI")

gui_widgets(x, reset = FALSE) <- value

guiWidgets(x, reset = FALSE) <- value

gui_ask(gui.or.name, ask)

guiAsk(gui.or.name, ask)
```

```
gui_ask(x) <- value

guiAsk(x) <- value
```

Arguments

gui.name	The name of the GUI. It is also the name of the object stored in SciViews:TempEnv where you can access it.
widgets	The list of widgets that GUI uses, listed in a priority order.
ask	Logical indicating if modal dialog boxes should be display (ask = TRUE), or if those dialog boxes are by-passed, using default values to simulate script running in non interactive mode, or to test scripts without interruption, using only provided default values (useful for automated tests).
reset	Should the GUI's main parameters (widgets, ask) be reset to default values?
gui	A gui object. If provided, it supersedes any value provided in gui.name.
x	A gui object.
value	The list of widgets to add to this GUI, in priority order, or should we change ask to TRUE, FALSE or NULL (then, use the default value stored in getOption("gui.ask")).
gui.or.name	A gui object or its name.

See Also

gui, setUI(), dont_ask()

Examples

```
# A 'gui' object named .GUI is automatically created in 'SciViews:TempEnv'
gui_list()

# Create a new GUI object to manage a separate GUI in the same R session
gui_add("myGUI")
gui_list()

# Change general properties of this GUI
gui_ask(myGUI) <- FALSE
# Add widgets to this GUI (you must provide methods for them)
# see the svDialogs package for examples
gui_widgets(myGUI) <- "tcltkWidgets"
gui_widgets(myGUI) # Added to existing ones if reset is FALSE

# Remove this new GUI
gui_remove("myGUI")
```

setUI *Set a property in the UI (User Interface), or start an action.*

Description

Using `setUI()` is the preferred way to set a property in a `gui` object. Similarly, `startUI()` should be used to indicate that an UI action requiring user input is initiated (say, a modal input or file selection dialog box).

Usage

```
setUI(..., gui = .GUI)

## S3 method for class 'gui'
setUI(fun, call, args, res, widgets, status, msg = NULL, ..., gui = .GUI)

startUI(..., gui = .GUI)

## S3 method for class 'gui'
startUI(
  fun,
  call,
  default,
  widgets = NULL,
  status = "busy-modal",
  msg = "Displaying a modal dialog box",
  msg.no.ask = "A modal dialog box was by-passed",
  ...,
  gui = .GUI
)
```

Arguments

`...`	Any other property of the GUI, provided as named arguments.
`gui`	A `gui` object.
`fun`	The name of the calling function. Only required if `call` is provided.
`call`	The call in the generic as obtained by `match.call()`.
`args`	A list with checked and/or reworked arguments for a method. The generic can do this work, so that code does not need to be duplicated in all its methods.
`res`	Any data returned by the GUI (the results).
`widgets`	The class name of the current widgets implementation.
`status`	Description of the current GUI status. Could be "ok", "busy", "busy-modal" (a modal dialog box is currently displayed), "by-passed" (the GUI was by-passed because `dont_ask()` returns `TRUE`), "error", or any other status indicator suitable for the current state of your GUI.

msg	The message that explains the status. Cannot be provided without status.
default	The default value to return if the UI is by-passed because in non interactive mode, or ask is FALSE.
msg.no.ask	The message that explains the status in case the UI is by-passed.

Methods (by class)

- setUI(gui): Set an UI property for a gui object.

Functions

- startUI(gui): Start an UI for a gui object.

See Also

gui_add(), $.gui()

Examples

```
# Imagine you implement a new input box
# In your function, you have this code:
myInput <- function(default = "an answer", gui = .GUI) {

  # Start a GUI action... or by-pass it!
  if (gui$startUI("myInput", call = match.call(), default = default,
    msg = "Displaying an input dialog box",
    msg.no.ask = "An input dialog box was by-passed")) {

    # Here the input dialog box is displayed and R waits for user feedback
    # ... [your code here]
    res <- "some results" # Imagine this is the text typed in the box

    # When the input dialog box is closed, the function should do:
    setUI(res = res, status = NULL)
  }
  invisible(gui)
}
```

Index

∗ **GUI API implementation**
  dont_ask, 4
  gui, 5
  gui_add, 6
  setUI, 8
∗ **misc**
  dont_ask, 4
  gui, 5
  gui_add, 6
  setUI, 8
$.gui *(gui)*, 5
$.gui(), *9*

dont_ask, 4
dont_ask(), *3*, *7*
dontAsk *(dont_ask)*, 4

gui, *4*, 5, *7*
gui_add, 6
gui_add(), *3*, *5*, *9*
gui_ask *(gui_add)*, 6
gui_ask(), *3*, *4*
gui_ask<- *(gui_add)*, 6
gui_change *(gui_add)*, 6
gui_change(), *3*
gui_list *(gui_add)*, 6
gui_list(), *3*
gui_remove *(gui_add)*, 6
gui_remove(), *3*
gui_widgets *(gui_add)*, 6
gui_widgets(), *3*
gui_widgets<- *(gui_add)*, 6
guiAdd *(gui_add)*, 6
guiAsk *(gui_add)*, 6
guiAsk<- *(gui_add)*, 6
guiChange *(gui_add)*, 6
guiList *(gui_add)*, 6
guiRemove *(gui_add)*, 6
guiWidgets *(gui_add)*, 6
guiWidgets<- *(gui_add)*, 6

is.gui *(gui)*, 5

print.gui *(gui)*, 5

setUI, 8
setUI(), *7*
startUI *(setUI)*, 8
svGUI *(svGUI-package)*, 2
svGUI-package, 2