Package 'manipulate'

July 22, 2025

Type Package

Title Interactive Plots for RStudio

Version 1.0.1

Date 2014-12-23

Maintainer JJ Allaire <jj@rstudio.com>

Description Interactive plotting functions for use within RStudio.
The manipulate function accepts a plotting expression and a set of controls (e.g. slider, picker, checkbox, or button) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

Depends R (>= 2.11.1)

SystemRequirements RStudio - http://www.rstudio.com/products/rstudio/

License GPL-2

LazyLoad yes

Author JJ Allaire [aut, cre] (R interface),
RStudio [cph]

NeedsCompilation no

Repository CRAN

Date/Publication 2014-12-24 01:18:44

Contents

manipulate-package 2
button 3
checkbox 4
isAvailable 5
manipulate 5
Manipulator Custom State 7
manipulatorMouseClick 8
picker 9
slider 10

Index **12**

`manipulate-package` *Interactive Plots for RStudio*

Description

Interactive plotting functions for use within RStudio.

Details

The `manipulate` function accepts a plotting expression and a set of controls (e.g. `slider`, `picker`, `checkbox`, or `button`) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

For example, to create a plot that enables manipulation of a parameter using a slider control you could use syntax like this:

```
manipulate(plot(1:x), x = slider(1, 10))
```

After this code is executed the plot is drawn using an initial value of 1 for `x`. A manipulator panel is also opened adjacent to the plot which contains a slider control used to change the value of `x` from 1 to 10.

Examples

```
## Not run:

## Create a plot with a manipulator
manipulate(plot(1:x), x = slider(5, 10))

## Using more than one slider
manipulate(
  plot(cars, xlim=c(x.min,x.max)),
  x.min=slider(0,15),
  x.max=slider(15,30))

## Filtering data with a picker
manipulate(
  barplot(as.matrix(longley[,factor]),
          beside = TRUE, main = factor),
  factor = picker("GNP", "Unemployed", "Employed"))

## Create a picker with labels
manipulate(
  plot(pressure, type = type),
  type = picker("points" = "p", "line" = "l", "step" = "s"))

## Toggle boxplot outlier display using checkbox
manipulate(
  boxplot(Freq ~ Class, data = Titanic, outline = outline),
```

```
  outline = checkbox(FALSE, "Show outliers"))

## Combining controls
manipulate(
  plot(cars, xlim = c(x.min, x.max), type = type,
       axes = axes, ann = label),
  x.min = slider(0,15),
  x.max = slider(15,30, initial = 25),
  type = picker("p", "l", "b", "c", "o", "h", "s", "S", "n"),
  axes = checkbox(TRUE, "Draw Axes"),
  label = checkbox(FALSE, "Draw Labels"))

## End(Not run)
```

---

button *Create a button control*

Description

Create a button control to enable triggering of conditional actions within manipulate expressions. When the user presses the button the manipulate expression will be executed with its associated value set to TRUE (in all other cases the value will be set to FALSE).

Usage

```
button(label)
```

Arguments

label	Label for button.

Value

An object of class "manipulator.button" which can be passed to the manipulate function.

See Also

manipulate, slider, picker, checkbox

Examples

```
## Not run:

## use a button to reset a random seed
manipulate(
  {
    if(resetSeed)
      set.seed(sample(1:1000))

    hist(rnorm(n=100, mean=0, sd=3), breaks=bins)
```

```
  },
  bins = slider(1, 20, step=1, initial =5, label="Bins"),
  resetSeed = button("Reset Seed")
)


## End(Not run)
```

---

checkbox *Create a checkbox control*

Description

Create a checkbox control to enable manipulation of logical plot variables.

Usage

```
checkbox(initial = FALSE, label = NULL)
```

Arguments

initial	Initial value for checkbox. Must be logical (defaults to FALSE).
label	Display label for checkbox. Defaults to the variable name if not specified.

Value

An object of class "manipulator.checkbox" which can be passed to the manipulate function.

See Also

manipulate, slider, picker, button

Examples

```
## Not run:

## Using checkboxes for boolean parameters
manipulate(
  plot(cars, axes = axes, ann = label),
  axes = checkbox(TRUE, "Draw Axes"),
  label = checkbox(FALSE, "Draw Labels"))

## Toggle boxplot outlier display using checkbox
manipulate(
  boxplot(Freq ~ Class, data = Titanic, outline = outline),
  outline = checkbox(FALSE, "Show outliers"))


## End(Not run)
```

isAvailable *Check whether manipulate is available*

Description

Check whether manipulate is available in the current front-end environment.

Usage

```
isAvailable()
```

Details

The manipulate package works only within the RStudio front-end.

Value

`TRUE` if manipulate is available, otherwise `FALSE`.

manipulate *Create an interactive plot*

Description

The `manipulate` function accepts a plotting expression and a set of controls (e.g. `slider`, `picker`, `checkbox`, or `button`) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

Usage

```
manipulate(`_expr`, ...)
```

Arguments

`_expr`	Expression to evalulate. The expression should result in the creation of a plot (e.g. `plot` or `qplot`). Note that the expression need not be a top-level plotting function, it could also be a custom function that creates a plot as part of its implementation. This expression will be re-evaluated with appropriate parameter substitution each time one of the manipulator control values is changed.
`...`	One or more named control arguments (i.e. `slider`, `picker`, `checkbox`, or `button`), or a list containing named controls.

Details

Once a set of manipulator controls are attached to a plot they remain attached and can be recalled whenever viewing the plot (a gear button is added to the top-left of the plot to indicate that it has a manipulator).

The `_expr` argument is evaluated using `withVisible`. If it's return value is visible then `print` is called. This enables manipulate expressions to behave simillarly to their being executed directly at the console.

The `_expr` argument uses a syntactially invalid (but backtick quoted) name to avoid clashes with named control arguments.

The `manipulatorSetState` and `manipulatorGetState` functions can be used to associate custom state with a manipulator (for example, to track the values used for previous plot executions). These values are stored in a custom environment which is stored along with the rest of the manipulator context.

Examples

```
## Not run:

## Create a plot with a manipulator
manipulate(plot(1:x), x = slider(5, 10))

## Using more than one slider
manipulate(
  plot(cars, xlim=c(x.min,x.max)),
  x.min=slider(0,15),
  x.max=slider(15,30))

## Filtering data with a picker
manipulate(
  barplot(as.matrix(longley[,factor]),
          beside = TRUE, main = factor),
  factor = picker("GNP", "Unemployed", "Employed"))

## Create a picker with labels
manipulate(
  plot(pressure, type = type),
  type = picker("points" = "p", "line" = "l", "step" = "s"))

## Toggle boxplot outlier display using checkbox
manipulate(
  boxplot(Freq ~ Class, data = Titanic, outline = outline),
  outline = checkbox(FALSE, "Show outliers"))

## Combining controls
manipulate(
  plot(cars, xlim = c(x.min, x.max), type = type,
       axes = axes, ann = label),
  x.min = slider(0,15),
  x.max = slider(15,30, initial = 25),
  type = picker("p", "l", "b", "c", "o", "h", "s", "S", "n"),
```

```
  axes = checkbox(TRUE, "Draw Axes"),
  label = checkbox(FALSE, "Draw Labels"))

## End(Not run)
```

Manipulator Custom State

Modify manipulator state

Description

These functions allow the storage of custom state variables across multiple evaluations of manipulator expressions. These functions are useful if the manipulate expression is a custom function (rather than a high level plotting function like plot) which requires reading and writing of persistent values.

Usage

```
manipulatorSetState(name, value)
manipulatorGetState(name)
```

Arguments

name	A chraracter string holding a state variable name.
value	An object holding a state value.

Value

manipulatorGetState returns a custom state value which was previously set by manipulatorSetState (or NULL if the specified name is not found).

See Also

manipulate

Examples

```
## Not run:

## set custom state variable
manipulatorSetState("last", x)

## get custom state variable
last <- manipulatorGetState("last")
if ( !is.null(last) ) {
  # do something interesting
}


## End(Not run)
```

`manipulatorMouseClick` *Receive notification of mouse clicks on a manipulator plot*

Description

This function can be called to determine if a mouse click on the plot was what caused the current invocation of the manipulate expression, and to determine the coordinates which were clicked.

Usage

```
manipulatorMouseClick()
```

Details

If a mouse click did occur, then the function returns a list with the coordinates which the user clicked on.

If a mouse click did not cause the current invocation of the manipulate expression (e.g. if it was caused by the user changing the value of a control) then the function returns NULL.

The mouse click coordinates are provided in device, user, and ndc coordinates. To convert these coordinates into other coordinate systems (e.g. cm or npc) you can use the `grconvertX` and `grconvertY` functions.

Note that the userX and userY coordinates are only applicable for base graphics plots (they are not applicable for grid, lattice, ggplot, etc). Therefore, for non-base graphics the userX and userY values will not contain valid coordinates.

Value

Returns a list containing the coordinates that user clicked (or NULL if a mouse click didn't occur):

`deviceX`	Device X coordinate (expressed in pixels)
`deviceY`	Device Y coordinate (expressed in pixels)
`userX`	User X coordinate (expressed in plot x units). Note that this value is only valid for base graphics.
`userY`	User Y coordinate (expressed in plot y units). Note that this value is only valid for base graphics.
`ndcX`	NDC X coordinate (0 to 1 from left to right)
`ndcY`	NDC Y coordinate (0 to 1 from bottom to top)

See Also

`manipulate`, `grconvertX`, `grconvertY`

picker *Create a picker control*

Description

Create a picker control to enable manipulation of plot variables based on a set of fixed choices.

Usage

```
picker(..., initial = NULL, label = NULL)
```

Arguments

...	Arguments containing objects to be presented as choices for the picker (or a list containing the choices). If an element is named then the name is used to display it within the picker. If an element is not named then it is displayed within the picker using as.character.
initial	Initial value for picker. Value must be present in the list of choices specified. If not specified defaults to the first choice.
label	Display label for picker. Defaults to the variable name if not specified.

Value

An object of class "manipulator.picker" which can be passed to the manipulate function.

See Also

manipulate, slider, checkbox, button

Examples

```
## Not run:

## Filtering data with a picker
manipulate(
  barplot(as.matrix(longley[,factor]),
          beside = TRUE, main = factor),
  factor = picker("GNP", "Unemployed", "Employed"))

## Create a picker with labels
manipulate(
  plot(pressure, type = type),
  type = picker("points" = "p", "line" = "l", "step" = "s"))

## Picker with groups
manipulate(
  barplot(as.matrix(mtcars[group,"mpg"]), beside=TRUE),
  group = picker("Group 1" = 1:11,
```

```
                "Group 2" = 12:22,
                "Group 3" = 23:32))

## Histogram w/ picker to select type
require(lattice)
require(stats)
manipulate(
  histogram(~ height | voice.part,
            data = singer, type = type),
  type = picker("percent", "count", "density"))


## End(Not run)
```

---

slider *Create a slider control*

Description

Create a slider control to allow manipulation of a plot variable along a numeric range.

Usage

```
slider(min, max, initial = min,
       label = NULL, step = NULL, ticks = TRUE)
```

Arguments

min	Minimum value for slider.
max	Maximum value for slider.
initial	Initial value for slider. Defaults to min if not specified.
label	Display label for slider. Defaults to the variable name if not specified.
step	Step value for slider. If not specified then defaults to 1 for integer ranges and single pixel granularity for floating point ranges (max - min divided by the number of pixels in the slider).
ticks	Show tick marks on the slider. Note that if the granularity of the step value is very low (more than 25 ticks would be shown) then ticks are automatically turned off.

Value

An object of class "manipulator.slider" which can be passed to the manipulate function.

See Also

manipulate, picker, checkbox, button

Examples

```
## Not run:

## Create a plot with a slider
manipulate(plot(1:x), x = slider(5, 10))

## Use multiple sliders
manipulate(
  plot(cars, xlim = c(x.min, x.max)),
  x.min = slider(0,15),
  x.max = slider(15,30))

## Specify a custom initial value for a slider
manipulate(
  barplot(1:x),
  x = slider(5, 25, initial = 10))

## Specify a custom label for a slider
manipulate(
  barplot(1:x),
  x = slider(5, 25, label = "Limit"))

## Specify a step value for a slider
manipulate(
  barplot(1:x),
  x = slider(5, 25, step = 5))

## End(Not run)
```

Index

∗ **dynamic**
    manipulate-package, 2
∗ **iplot**
    manipulate-package, 2
∗ **package**
    manipulate-package, 2

as.character, *9*

button, *2*, 3, *4, 5*, *9, 10*

checkbox, *2, 3*, 4, *5*, *9, 10*

grconvertX, *8*
grconvertY, *8*

isAvailable, 5

manipulate, *2–5*, 5, *7–10*
manipulate-package, 2
Manipulator Custom State, 7
manipulatorGetState, *6*
manipulatorGetState *(*Manipulator Custom State*)*, 7
manipulatorMouseClick, 8
manipulatorSetState, *6*
manipulatorSetState *(*Manipulator Custom State*)*, 7

picker, *2–5*, 9, *10*
plot, *7*
print, *6*

slider, *2–5*, *9*, 10

withVisible, *6*