

# Package ‘formatR’

January 2, 2012

**Type** Package

**Title** Format R Code Automatically

**Version** 0.3-4

**Date** 2011-11-21

**Author** Yihui Xie

**Maintainer** Yihui Xie <xie@yihui.name>

**Description** This package provides a function `tidy.source()` and optionally a GUI (by `gWidgets` and preferably `gWidgetsRGtk2`) to format R source code. Spaces and indent will be added to the code automatically, and comments will be preserved under certain conditions, so that R code will be more human-readable and tidy.

**Imports** parser (>= 0.0-14)

**Suggests** gWidgetsRGtk2

**License** GPL-2 | GPL-3

**URL** <https://github.com/yihui/formatR/wiki>

**BugReports** <https://github.com/yihui/formatR/issues>

**Collate** ‘tidy.R’ ‘GUI.R’ ‘usage.R’ ‘zzz.R’ ‘eval.R’

**Repository** CRAN

**Date/Publication** 2011-11-21 08:06:20

## R topics documented:

<code>tidy.dir</code> . . . . .	2
<code>tidy.eval</code> . . . . .	3
<code>tidy.gui</code> . . . . .	4
<code>tidy.source</code> . . . . .	5
<code>usage</code> . . . . .	9

---

tidy.dir	<i>Format the R scripts under a directory</i>
----------	---

---

### Description

This function first looks for all the R scripts under a directory (using the pattern "\\.[RrSsQq]\$"), then uses [tidy.source](#) to tidy these scripts. The original scripts will be overwritten with reformatted code if reformatting was successful. You may need to back up the original directory first if you do not fully understand the tricks [tidy.source](#) is using.

### Usage

```
tidy.dir(path = ".", recursive = FALSE, ...)
```

### Arguments

path	the directory
recursive	whether to recursively look for R scripts under path
...	other arguments to be passed to <a href="#">tidy.source</a>

### Value

NULL

### Author(s)

Yihui Xie <<http://yihui.name>>

### See Also

[tidy.source](#)

### Examples

```
library(formatR)

path = tempdir()
file.copy(system.file("demo", package = "base"), path,
          recursive = TRUE)
tidy.dir(path, recursive = TRUE)
```

---

`tidy.eval`*Evaluate R code and mask the output by a prefix*

---

## Description

This function is designed to insert the output of each chunk of R code into the source code without really breaking the source code, since the output is masked in comments.

## Usage

```
tidy.eval(source = "clipboard", ..., file = "", prefix = "## ",
  envir = .GlobalEnv)
```

## Arguments

<code>source</code>	the input filename (by default the clipboard; see <a href="#">tidy.source</a> )
<code>...</code>	other arguments passed to <a href="#">tidy.source</a>
<code>file</code>	the file to write by <a href="#">cat</a> ; by default the output is printed on screen
<code>prefix</code>	the prefix to mask the output
<code>envir</code>	the environment in which to evaluate the code (by default the global environment; if we do not want to mess up with the global environment, we can set <code>envir = NULL</code> or <code>envir = new.env()</code> )

## Value

Evaluated R code with corresponding output (printed on screen or written in a file).

## Author(s)

Yihui Xie <<http://yihui.name>>

## References

<https://github.com/yihui/formatR/wiki/>

## Examples

```
library(formatR)
## evaluate simple code as a character vector
tidy.eval(text = c("a<-1+1;a", "matrix(rnorm(10),5)"))

## evaluate a file
tidy.eval(source = file.path(system.file(package = "stats"),
  "demo", "nlm.R"), keep.blank.line = TRUE)
```

---

`tidy.gui`*A GUI to format R code*

---

## Description

Create a GUI (via GTK+ by default) to format R code.

## Usage

```
tidy.gui(guiToolkit = "RGtk2")
```

## Arguments

`guiToolkit`      the GUI toolkit to use

## Details

This function calls [tidy.source](#) to format R code. Spaces and indent will be added to the code automatically.

We can either open an R source file or directly write R code in the text widget. Click the “convert” button, and the code will become tidy. See [tidy.source](#) for more details.

## Value

the text widget is returned

## Note

By default, the interface is based on GTK+ (R package **RGtk2**), but other options (**tcltk**, **rJava** and **Qt**) are possible too. See the examples below. Note the “Font” button is only for the GTK+ interface.

## Author(s)

Yihui Xie <<http://yihui.name>>

## References

<https://github.com/yihui/formatR/wiki/> (screenshots)

## See Also

[tidy.source](#)

**Examples**

```
## Not run:
library("gWidgetsRGtk2")

## a GUI will show up on loading if one of the gWidgets
## toolkit is present (e.g. via library(gWidgetsRGtk2))
library(formatR)

g = tidy.gui()

## we have control over the text widget, e.g. set or get
# the text

svalue(g) = c("# a single line of comments is preserved",
  "1+1", "if(TRUE){", paste("x=1 ", "# inline comments!"),
  "}else{", "x=2;print('Oh no... ask the right bracket to go away!')}",
  "1*3 # another inline comment")

## click 'Convert' now, and see

cat(svalue(g), sep = "\n") # get its value

## tcl/tk interface: need gWidgetstcltk package
tidy.gui("tcltk")

## End(Not run)
```

tidy.source

*'Tidy up' R code while preserving comments***Description**

This function has nothing to do with code optimization; it just returns parsed source code, but also tries to preserve comments, which is different with [parse](#). See 'Details'.

**Usage**

```
tidy.source(source = "clipboard", keep.comment = getOption("keep.comment",
  TRUE), keep.blank.line = getOption("keep.blank.line", TRUE),
  keep.space = getOption("keep.space", FALSE), replace.assign = getOption("replace.assign",
  FALSE), output = TRUE, text = NULL, width.cutoff = 0.75 *
  getOption("width"), ...)
```

**Arguments**

source            a character string: location of the source code (default to be the clipboard; this means we can copy the code to clipboard and use `tidy.source()` without specifying the argument `source`)

keep.comment	whether to keep comments (TRUE by default)
keep.blank.line	whether to keep blank lines (TRUE by default)
keep.space	whether to preserve the leading spaces in the single lines of comments (default FALSE)
replace.assign	whether to replace the assign operator = with <-
output	output to the console or a file using <code>cat</code> ?
text	an alternative way to specify the input: if it is NULL, the function will read the source code from the source argument; alternatively, if text is a character vector containing the source code, it will be used as the input and the source argument will be ignored
width.cutoff	passed to <code>deparse</code> : integer in [20, 500] determining the cutoff at which line-breaking is tried (default to be $0.75 * \text{getOption("width")}$ )
...	other arguments passed to <code>cat</code> , e.g. file (this can be useful for batch-processing R scripts, e.g. <code>tidy.source(source = 'input.R', file = 'output.R')</code> )

## Details

This function helps the users to tidy up their source code in a sense that necessary indents and spaces will be added, but comments will be preserved if `keep.comment = TRUE`.

The method to preserve comments is to protect them as strings in disguised assignments. For example, there is a single line of comments in the source code:

```
# asdf
```

It will be first masked as

```
.IDENTIFIER1 <- " # asdf.IDENTIFIER2"
```

which is a legal R expression, so `parse` can deal with it and will no longer remove the disguised comments. In the end the identifiers will be removed to restore the original comments, i.e. the strings `'.IDENTIFIER1 <- ''` and `'.IDENTIFIER2''` are replaced with empty strings.

“Inline” comments are handled differently: two spaces will be added before the hash symbol #, e.g.

```
1+1# comments
```

will become

```
1+1 # comments
```

Inline comments are first disguised as a weird operation with its preceding R code, which is essentially meaningless but syntactically correct! For example,

```
1+1 %InLiNe_IdEnTiFiEr% "# comments"
```

then `parse` will deal with this expression; again, the disguised comments will not be removed. In the end, inline comments will be freed as well (remove the operator `%InLiNe_IdEnTiFiEr%` and surrounding double quotes).

All these special treatments to comments are due to the fact that `parse` and `deparse` can tidy the R code at the price of dropping all the comments.

**Value**

A list with components

<code>text.tidy</code>	the reformatted code as a character vector
<code>text.mask</code>	the code containing comments, which are masked in assignments or with the weird operator
<code>begin.comment, end.comment</code>	identifiers used to mark the comments

**Note**

When `keep.comment == TRUE`, *all your double quotes in the comments will be replaced by single quotes!!* For example,

```
1 + 1 # here is "comment"
```

will become

```
1 + 1 # here is 'comment'
```

There are global options which can override some arguments: the argument `keep.comment` gets its value from `options('keep.comment')` by default; `keep.blank.line` from `options('keep.blank.line')`, `keep.space` from `options('keep.space')`, and `replace.assign` from `options('replace.assign')`. If these options are NULL, the default values will be TRUE, TRUE, FALSE and FALSE respectively.

Also note that if `keep.space` is FALSE, single lines of long comments will be wrapped into shorter ones automatically. Otherwise, long comments will not be wrapped, so they may exceed the page margin, and `\\t` will be replaced with `\\t`. Roxygen comments will not be wrapped in any case.

Be sure to read the reference to know other limitations.

**Author(s)**

Yihui Xie <<http://yihui.name>> with substantial contribution from Yixuan Qiu <<http://yixuan.cos.name>>

**References**

<https://github.com/yihui/formatR/wiki/> (an introduction to this package, with examples and further notes)

The package vignette also contains some examples (see `vignette('formatR', package = 'formatR')`).

**See Also**

[parse](#), [deparse](#), [cat](#)



```
## if you've copied R code into the clipboard
if (interactive()) {
  tidy.source("clipboard")
  ## write into clipboard again
  tidy.source("clipboard", file = "clipboard")
}

## the if-else structure
tidy.source(text = c("{if(TRUE)1 else 2; if(FALSE){1+1",
  "## comments", "} else 2}"))
```

---

usage

*Show the usage of a function*

---

## Description

Print the reformatted usage of a function. The arguments of the function are searched by [argsAnywhere](#), so the function can be either exported or non-exported in a package. S3 methods will be marked.

## Usage

```
usage(FUN, width = 0.77)
```

## Arguments

FUN	the function name
width	the width of output

## Value

NULL; the usage is printed on screen

## Author(s)

Yihui Xie <<http://yihui.name>>

## See Also

[tidy.source](#)

## Examples

```
library(formatR)
usage(var)

usage(plot)

usage(plot.default) # default method
```

```
usage(plot.lm) # on the 'lm' class  
usage(usage)  
usage(barplot.default, width = 0.6) # narrower output
```

# Index

## \*Topic **IO**

tidy.source, [5](#)

argsAnywhere, [9](#)

cat, [3](#), [6](#), [7](#)

deparse, [6](#), [7](#)

parse, [5–7](#)

tidy.dir, [2](#)

tidy.eval, [3](#)

tidy.gui, [4](#)

tidy.source, [2–4](#), [5](#), [9](#)

usage, [9](#)