

# Package ‘prettifyAddins’

July 19, 2022

**Type** Package

**Title** 'RStudio' Addins to Prettify 'JavaScript', 'C++', 'Python', and More

**Version** 2.5.0

**Description** Provides 'RStudio' addins to prettify 'HTML', 'CSS', 'SCSS', 'JavaScript', 'JSX', 'Markdown', 'C(++)', 'LaTeX', 'Python', 'Julia', 'XML', 'Java', 'JSON', 'Ruby', and to reindent 'C(++)', 'Fortran', 'Java', 'Julia', 'Python', 'SAS', 'Scala', 'Shell', 'SQL' and ``TypeScript``. Two kinds of addins are provided: 'Prettify' and 'Indent'. The 'Indent' addins only reindent the code, while the 'Prettify' addins also modify the code, e.g. trailing semi-colons are added to 'JavaScript' code when they are missing.

**License** GPL-3

**URL** <https://github.com/stla/prettifyAddins>

**BugReports** <https://github.com/stla/prettifyAddins/issues>

**Imports** httr, rstudioapi, shiny, tools, utils, webdriver, xml2, XRJulia

**Suggests** miniUI, shinyAce, shinythemes, testthat, V8

**Encoding** UTF-8

**RoxygenNote** 7.2.0

**NeedsCompilation** no

**Author** Stéphane Laurent [aut, cre],  
James Long and contributors [cph] ('Prettier' library),  
Zeb Zhao [cph] ('indent.js' library),  
Marijn Haverbeke [cph] ('CodeMirror' library),  
George Leslie-Waksman and other contributors [cph] ('sql-formatter'  
library),  
Austin Cheney [cph] ('prettydiff' library),  
John Schlinkert [ctb, cph] (word-wrap)

**Maintainer** Stéphane Laurent <laurent\_step@outlook.fr>

**Repository** CRAN

**Date/Publication** 2022-07-19 12:00:06 UTC

**R topics documented:**

getPrettifiableLanguages . . . . .	2
prettifyAddins . . . . .	2
prettifyAddins-imports . . . . .	3
prettifyCLANG . . . . .	3
prettifyHTML . . . . .	4
prettifyJulia . . . . .	4
prettifyLaTeX . . . . .	5
prettifyPython . . . . .	6
prettifyXML . . . . .	6
prettify_FCA . . . . .	7
prettify_Shiny . . . . .	8
prettify_V8 . . . . .	9
reindent_PhantomJS . . . . .	10
reindent_Shiny . . . . .	11
reindent_V8 . . . . .	11
wordWrap . . . . .	12

<b>Index</b>	<b>13</b>
--------------	-----------

---

getPrettifiableLanguages

*Prettifiable languages*

---

**Description**

Returns the list of languages that are supported by this package.

**Usage**

```
getPrettifiableLanguages()
```

---

prettifyAddins

*Prettify Addins*

---

**Description**

This package provides some RStudio addins: Prettify addins and Indent addins. To run an addin, select it from the Addins menu within RStudio. The Indent addins only reindent the code, while the Prettify addins also modify the code, e.g. they add trailing semi-colons to JavaScript code when they are missing.

**Examples**

```
# get the list of supported languages:
getPrettifiableLanguages()
```

---

prettifyAddins-imports

*Install PhantomJS*

---

### Description

This function is imported from the 'webdriver' package. Follow the link to its documentation: [install\\_phantomjs](#)

---

prettifyCLANG

*Prettify C, C++, Java*

---

### Description

Prettify some C, C++ or Java code.

### Usage

```
prettifyCLANG(contents = NA, language = NA, tabSize = NULL)
```

### Arguments

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code; when the contents is read from a file, this option is ignored, because the language is obtained from the extension of the file
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents is read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

### Value

The pretty code in a character string.

### Note

This function requires the command line utility `clang-format`.

---

```
prettifyHTML      Prettify HTML
```

---

**Description**

Prettify some HTML code. It works for big files.

**Usage**

```
prettifyHTML(contents = NA, tabSize = NULL)
```

**Arguments**

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents is read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

**Value**

The pretty code in a character string.

**Note**

This function requires the command line utility prettydiff, to install with npm.

---

```
prettifyJulia     Prettify Julia
```

---

**Description**

Prettify Julia code.

**Usage**

```
prettifyJulia(contents = NA, tabSize = NULL)
```

**Arguments**

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents is read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

**Value**

The pretty code in a character string.

**Note**

This function requires that Julia is installed on your system and that the Julia package `JuliaFormatter` is installed.

---

```
prettifyLaTeX
```

```
Prettify LaTeX
```

---

**Description**

Prettify LaTeX code, including Sweave code, sty files, cls files, and bib files.

**Usage**

```
prettifyLaTeX(contents = NA, tabSize = NULL, log = FALSE)
```

**Arguments**

<code>contents</code>	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
<code>tabSize</code>	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents is read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2
<code>log</code>	logical, whether to generate a log file (it will be named <code>indent.log</code> )

**Value**

The pretty code in a character string.

**Note**

This function requires the command line utility `latexindent`.

---

prettifyPython	<i>Prettify Python</i>
----------------	------------------------

---

**Description**

Prettify Python code.

**Usage**

```
prettifyPython(contents = NA)
```

**Arguments**

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
----------	--

**Value**

The pretty code in a character string.

**Note**

This function requires [black](#).

---

prettifyXML	<i>Prettify XML</i>
-------------	---------------------

---

**Description**

Prettify some XML or SVG code.

**Usage**

```
prettifyXML(contents = NA, tabSize = NULL)
```

**Arguments**

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

**Details**

The code is prettified with the help of the command line utility `xmlint` if it is available, otherwise the `xml2` is used.

**Value**

The pretty code in a character string.

---

prettify_FCA	<i>Prettify Java, JSON or Ruby</i>
--------------	------------------------------------

---

**Description**

Prettify Java code, JSON code or Ruby code.

**Usage**

```
prettify_FCA(contents = NA, language = NA)
```

**Arguments**

<code>contents</code>	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
<code>language</code>	the language of the code, such as "json"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and <code>language=NA</code> , then the language is guessed from the file extension

**Value**

The pretty code in a character string.

**Note**

This function requires a connection to Internet.

**Examples**

```
library(prettifyAddins)

code <- c(
  "{a: [0,1, 2 ]}",
  "f: function( x){return x+1}" # this function will be prettified too
)

## Not run:
cat(prettify_FCA(code, "json"))
## End(Not run)
```

---

prettify\_Shiny      *Prettify code using Shiny*

---

### Description

Prettify some code using a Shiny app.

### Usage

```
prettify_Shiny(contents = NA, language = NA, tabSize = NULL, themeInfo = NULL)
```

### Arguments

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code, such as "javascript" or "JavaScript"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and language=NA, then the language is guessed from the file extension
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2
themeInfo	this argument is not important, it controls the theme of the Shiny app; it must be NULL or a list with two fields: editor, the name of a theme, and dark, a logical value, which tells whether the theme is dark

### Value

The pretty code in a character string.

### Examples

```
library(prettifyAddins)

code <- c(
  "function f(x){",
  "  return x+1",
  "}"
)
if(interactive()){
  cat(prettify_Shiny(code, "javascript"))
}
```



---

prettify_V8	<i>Prettify code using V8</i>
-------------	-------------------------------

---

## Description

Prettify some code using the V8 package.

## Usage

```
prettify_V8(contents = NA, language = NA, tabSize = NULL)
```

## Arguments

contents	the code to be prettified; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code, such as "javascript"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and language=NA, then the language is guessed from the file extension
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

## Value

The pretty code in a character string.

## Examples

```
library(prettifyAddins)

code <- c(
  "function f(x){",
  "  return x+1",
  "}"
)
cat(prettify_V8(code, "JavaScript"))
```

reindent\_PhantomJS     *Reindent code using PhantomJS*

---

### Description

Reindent some code using PhantomJS.

### Usage

```
reindent_PhantomJS(contents = NA, language = NA, tabSize = NULL)
```

### Arguments

contents	the code to be reindented; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code, such as "python"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and language=NA, then the language is guessed from the file extension
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

### Value

The reindented code in a character string.

### Note

This function requires the 'phantomjs' command-line utility.

### Examples

```
library(prettifyAddins)

code <- c(
  'if test == 1:',
  'print "it is one"',
  'else:',
  'print "it is not one"'
)

## Not run:
cat(reindent_PhantomJS(code, "python"))
## End(Not run)
```

---

reindent_Shiny	<i>Reindent code using Shiny</i>
----------------	----------------------------------

---

**Description**

Reindent some code using a Shiny app.

**Usage**

```
reindent_Shiny(contents = NA, language = NA, tabSize = NULL, themeInfo = NULL)
```

**Arguments**

contents	the code to be reindented; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code, such as "javascript"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and language=NA, then the language is guessed from the file extension
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2
themeInfo	this argument is not important, it controls the theme of the Shiny app; it must be NULL or a list with two fields: editor, the name of a theme, and dark, a logical value, which tells whether the theme is dark

**Value**

The reindented code in a character string.

---

reindent_V8	<i>Reindent code using V8</i>
-------------	-------------------------------

---

**Description**

Reindent some code using the V8 package.

**Usage**

```
reindent_V8(contents = NA, language = NA, tabSize = NULL)
```

**Arguments**

contents	the code to be reindented; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
language	the language of the code, such as "javascript"; see <a href="#">getPrettifiableLanguages</a> ; if the contents are read from a file and language=NA, then the language is guessed from the file extension
tabSize	number of spaces of the indentation (usually 2 or 4); if NULL (the default), there are two possibilities: if the contents are read from the current file in RStudio, then the number of spaces will be the one you use in RStudio; otherwise it is set to 2

**Value**

The reindented code in a character string.

**Examples**

```
library(prettifyAddins)

code <- c(
  "function f(x){",
  "  return x+1",
  "}"
)
cat(reindent_V8(code, "javascript"))
```

---

wordWrap

*Word wrap using V8*


---

**Description**

Word wrap a text.

**Usage**

```
wordWrap(contents = NA, ncharacters = 80)
```

**Arguments**

contents	the text to be wrapped; there are three possibilities for this argument: NA (default), to use the file currently opened in RStudio; the path to a file; or the code given as a character vector
ncharacters	target number of characters per line

**Value**

The wrapped text in a character string.

# Index

[getPrettifiableLanguages](#), [2](#), [7–12](#)

[install\\_phantomjs](#), [3](#)  
[install\\_phantomjs](#)  
    ([prettifyAddins-imports](#)), [3](#)

[prettify\\_FCA](#), [7](#)  
[prettify\\_Shiny](#), [8](#)  
[prettify\\_V8](#), [9](#)  
[prettifyAddins](#), [2](#)  
[prettifyAddins-imports](#), [3](#)  
[prettifyCLANG](#), [3](#)  
[prettifyHTML](#), [4](#)  
[prettifyJulia](#), [4](#)  
[prettifyLaTeX](#), [5](#)  
[prettifyPython](#), [6](#)  
[prettifyXML](#), [6](#)

[reindent\\_PhantomJS](#), [10](#)  
[reindent\\_Shiny](#), [11](#)  
[reindent\\_V8](#), [11](#)

[wordWrap](#), [12](#)