Package 'cppcheckR'

June 10, 2022

Type Package

Title Check 'C' and 'C++' Files using 'Cppcheck'

Version 1.0.0

Description Allow to run 'Cppcheck' (<https://cppcheck.sourceforge.io/>) on 'C' and 'C++' files with a 'R' command or a 'RStudio' addin. The report appears in the 'RStudio' viewer pane as a formatted 'HTML' file. It is also possible to get this report with a 'shiny' application. 'Cppcheck' can spot many error types and it can also give some recommendations on the code.

License GPL-3

URL https://github.com/stla/cppcheckR

BugReports https://github.com/stla/cppcheckR/issues

Imports htmlwidgets, rmarkdown, rstudioapi, shiny, utils, V8, xml2

Suggests htmltools, shinyAce, shinybusy, shinyFiles, shinyjqui, shinyWidgets

Encoding UTF-8

RoxygenNote 7.2.0

SystemRequirements cppcheck

NeedsCompilation no

Author Stéphane Laurent [aut, cre], Amit Gupta [cph] ('fast-xml-parser' library), luyilin [cph] ('json-format-highlight' library)

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Repository CRAN

Date/Publication 2022-06-10 09:10:09 UTC

R topics documented:

`cppcheckR` *Check a C/C++ file or a folder*

Description

HTML widget which runs **Cppcheck**.

Usage

```
cppcheckR(
  path,
  std = NULL,
  def = NULL,
  undef = NULL,
  checkConfig = FALSE,
  height = NULL,
  elementId = NULL
)
```

Arguments

`path`	path to a C/C++ file or to a folder containing C/C++ files
`std`	the standard, one of `"c89"`, `"c99"`, `"c11"`, `"c++03"`, `"c++11"`, `"c++14"`, `"c++17"`, `"c++20"`; if `NULL`, you will be prompted to select it
`def`	character vector of symbols you want to define, e.g. `"__cplusplus"` or `"DEBUG=1"`; if `NULL`, you will be prompted to enter them; set `def=NA` if you don't want to define any symbol
`undef`	character vector of symbols you want to undefine; if `NULL`, you will be prompted to enter them; set `undef=NA` if you don't want to undefine any symbol
`checkConfig`	Boolean, whether to run **Cppcheck** with the option `--check-config`; this tells you which header files are missing
`height`	height in pixels (defaults to automatic sizing)
`elementId`	an id for the widget, this is usually useless

Value

A `htmlwidget` object.

Examples

```
example <- function(file){
  filepath <- system.file("cppexamples", file, package = "cppcheckR")
  lines <- readLines(filepath)
  print(cppcheckR(filepath, std = "c++03", def = NA, undef = NA))
  message(file, ":")
  cat(paste0(format(seq_along(lines)), ". ", lines), sep = "\n")
}
example("memleak.cpp")
example("outofbounds.cpp")
example("unusedvar.cpp")
example("useless.cpp")
```

---

cppcheckR-shiny *Shiny bindings for* cppcheckR

Description

Output and render functions for using cppcheckR within Shiny applications and interactive Rmd documents.

Usage

```
cppcheckROutput(outputId, width = "100%", height = "400px")

renderCppcheckR(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended
expr	an expression that generates a 'cppcheckR' widget
env	the environment in which to evaluate expr
quoted	logical, whether expr is a quoted expression (with quote())

Value

cppcheckROutput returns an output element that can be included in a Shiny UI definition, and renderCppcheckRR returns a shiny.render.function object that can be included in a Shiny server definition.

`json2html` *JSON to HTML*

Description

Render a formatted JSON string in HTML.

Usage

```
json2html(
  json,
  outfile = NULL,
  pandoc = FALSE,
  style = paste0("background-color: #051C55; color: #E76900; ",
    "font-size: 15px; font-weight: bold; margin: 0; ",
    "white-space: pre-wrap; outline: #051C55 solid 10px;"),
  keyColor = "crimson",
  numberColor = "chartreuse",
  stringColor = "lightcoral",
  trueColor = "#00cc00",
  falseColor = "#ff8080",
  nullColor = "cornflowerblue"
)
```

Arguments

`json`	a JSON string or a path to a JSON file
`outfile`	either a path to a html file, or `NULL` if you don't want to write the output to a file
`pandoc`	Boolean, whether to use pandoc
`style`	some CSS for the container, only if `pandoc=FALSE`
`keyColor`	color of the keys, only if `pandoc=FALSE`
`numberColor`	color of the numbers, only if `pandoc=FALSE`
`stringColor`	color of the strings, only if `pandoc=FALSE`
`trueColor`	color of the `true` keyword, only if `pandoc=FALSE`
`falseColor`	color of the `false` keyword, only if `pandoc=FALSE`
`nullColor`	color of the `null` keyword, only if `pandoc=FALSE`

Value

Nothing if `outfile` is not `NULL`, otherwise the HtML as a character string.

Examples

```
library(cppcheckR)
xml <- system.file("extdata", "order-schema.xml", package = "xml2")
json <- xml2json(xml)
html <- json2html(json)
library(htmltools)
if(interactive()){
  browsable(HTML(html))
}
# with pandoc
html <- json2html(json, pandoc = TRUE)
if(interactive()){
  browsable(HTML(html))
}
```

---

shinyCppcheck *Shiny application to check C/C++*

Description

Run a shiny application to check C/C++ files.

Usage

```
shinyCppcheck()
```

Value

Nothing, this just launches a Shiny app.

Note

The packages listed in the **Suggests** field of the package description are required.

---

xml2json *XML to JSON*

Description

Convert XML to a JSON string.

Usage

```
xml2json(
  xml,
  spaces = 2L,
  linebreaks = FALSE,
  replacer = NULL,
  attributeNamePrefix = "@_",
  textNodeName = "#text",
  ignoreAttributes = FALSE,
  ignoreNameSpace = FALSE,
  parseNodeValue = TRUE,
  parseAttributeValue = TRUE,
  trimValues = TRUE
)
```

Arguments

`xml`	either a XML file or some XML given as a character string
`spaces`	an integer, the indentation
`linebreaks`	Boolean, whether to break the lines in the JSON string when there are some linebreaks; this generates an invalid JSON string
`replacer`	character vector, the body of the second argument of `JSON.stringify` (the replacer function), or `NULL`
`attributeNamePrefix`	prefix for the attributes
`textNodeName`	name of text nodes, which appear for nodes which have both a text content and some attributes
`ignoreAttributes`	Boolean, whether to ignore the attributes
`ignoreNameSpace`	Boolean, whether to ignore the namespaces
`parseNodeValue`	Boolean, whether to parse the node values to numbers if possible
`parseAttributeValue`	Boolean, whether to parse the attribute values to numbers if possible
`trimValues`	Boolean, whether to trim the values

Value

A JSON string.

Examples

```
xml <- system.file("extdata", "order-schema.xml", package = "xml2")
cat(xml2json(xml))
#
js <- c(
```

```
  'if(key === "@_type"){',
  '  return undefined;',
  '} else if(key === "@_name"){',
  '  return value.toUpperCase();',
  '}',
  'return value;'
)
cat(xml2json(xml, linebreaks = TRUE, replacer = js))
```

Index