

# Package ‘cppcheckR’

May 8, 2026

**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

## Contents

cppcheckR . . . . .	2
cppcheckR-shiny . . . . .	3

json2html . . . . .	4
shinyCppcheck . . . . .	5
xml2json . . . . .	5

<b>Index</b>	<b>8</b>
--------------	----------

---

cppcheckR	<i>Check a C/C++ file or a folder</i>
-----------	---------------------------------------

---

## 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 <b>Cppcheck</b> with the option <code>--check-config</code> ; 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 <code>'cppcheckR'</code> widget
env	the environment in which to evaluate expr
quoted	logical, whether expr is a quoted expression (with <code>quote()</code> )

## 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**

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

**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	<i>Shiny application to check C/C++</i>
---------------	-----------------------------------------

---

## 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	<i>XML to JSON</i>
----------	--------------------

---

## 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**

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

`cppcheckR`, [2](#), [3](#)  
`cppcheckR-shiny`, [3](#)  
`cppcheckROutput` (`cppcheckR-shiny`), [3](#)  
`json2html`, [4](#)  
`renderCppcheckR` (`cppcheckR-shiny`), [3](#)  
`shinyCppcheck`, [5](#)  
`xml2json`, [5](#)