Package: cssparser (via r-universe)

May 25, 2026

Type Package

Title Simple CSS Parser and Tools

Version 0.1.1

Author mikefc

Maintainer mikefc <mikefc@coolbutuseless.com>

Description Simple CSS parser and tools.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

URL https://github.com/coolbutuseless/cssparser,
https://coolbutuseless.github.io/package/cssparser

Imports stringi, xml2, selectr

Suggests rmarkdown, knitr, testthat (>= 3.0.0), gridtext, jpeg

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 2.10)

Config/pak/sysreqs libicu-dev libxml2-dev

Repository https://trevorld.r-universe.dev

Date/Publication 2021-11-02 09:25:55 UTC

RemoteUrl https://github.com/coolbutuseless/cssparser

RemoteRef HEAD

RemoteSha a5a3765fd8b058ae82790045b7b2443f5cbf4d8b

Contents

alpha_mul *Multiply alpha channel by given value*

Description

Multiply alpha channel by given value

Usage

```
alpha_mul(x, alpha = 1)
```

Arguments

x	hex colour with alpha with total of 8 hex digits
alpha	alpha [0, 1]

alpha_set *Set alpha channel*

Description

Set alpha channel

Usage

```
alpha_set(x, alpha = 1)
```

Arguments

x	hex colour with alpha with total of 8 hex digits
alpha	alpha [0, 1]

cat_indent *A version of cat which has an indentation depth*

Description

A version of cat which has an indentation depth

Usage

```
cat_indent(depth, ..., sep = "")
```

Arguments

depth	the indentation depth
...	arguments passed to cat()
sep	argument passed to cat()

css_apply *Create a named list (indexed by* xpath*) of the final computed style for each element*

Description

Create a named list (indexed by xpath) of the final computed style for each element

Usage

```
css_apply(xml, css = list(), svg = FALSE)
```

Arguments

xml	xml document as returned by `xml2::read_html()` or `xml2::read_xml()`
css	list of rules parsed from CSS stylesheet. E.g as returned by `cssparser::read_css()`
svg	include SVG presentation tags in the cascade? default: FALSE

Value

named list of styles where the name is the xpath to a node in the document, and the value is a named list of property/values for this element.

css_apply_inline — *Apply the CSS to the given HTML, storing the result as inline 'style' tags on each element*

Description

Apply the CSS to the given HTML, storing the result as inline 'style' tags on each element

Usage

```
css_apply_inline(xml, css)
```

Arguments

xml	html/xml document (as read by `xml2::read_xml()`)
css	CSS stylesheet to apply (as read by `cssparser::read_css()`)

Value

new xml document with final computed CSS written to inline style attribute on each element.

css_colour_to_hex — *Convert CSS colour to standard hex colour (with alpha0)*

Description

CSS colours can be lots of things:

Usage

```
css_colour_to_hex(x)
```

Arguments

x	CSS colour

Details

- Hex colour with 3, 6 or 8 characters
- CSS colour name e.g. 'red'. Not all CSS colours correspond to R colours e.g. 'silver' is not in R
- rgb(), hsl(), hcl(), lab() and other colourspace-specific colour constructores. Only rgb and hsl are currently handled
- color() for complex colourspace specifidation. Not handled yet

- 'transparent' and sometimes 'none' to indicate '#00000000'
- 'currentColor' or 'currentcolor' to indicate the colour for this element should be taken from whatever the current 'color' property is
- for SVG, colours colours can be references to 'linearGradient' and 'radialGradient' specifications, or patterns

Ref: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#currentcolor_keyword

Value

hex colour (always with alpha channel)

`css_length` *Create a new CSS length object*

Description

Create a new CSS length object

Usage

```
css_length(x, unit)
```

Arguments

`x`	numeric value
`unit`	character string for unit

`css_length_as_pixels` *Convert a CSS unit into a rough pixel measurement*

Description

This is a naive conversion.

Usage

```
css_length_as_pixels(x, font_size = 12, ...)
```

Arguments

`x`	object of type `css_length` as returned by `css_string_to_css_length()`
`font_size`	font size to calculate 'em' with. default: 12
`...`	other arguments ignored

Details

For a proper conversion, it would have to take into account the font-size on the root element (for `rem` units), and various things to do with framepoint and viewing size of the element and its parents.

Value

simple numeric value

`css_merge` *Merge multiple stylesheets given in priority order*

Description

Merge multiple stylesheets given in priority order

Usage

```
css_merge(...)
```

Arguments

`...`	multiple CSS objects. The order in which these arguments are given reflect the priority of the stylesheets from lowest to highest priority. Later stylesheets (i.e. high priority) will override any styles declared earlier (lower priority)

Value

final cascaded stylesheet

Examples

```
## Not run:
css1 <- read_css("chrome_builtin.css")
css2 <- read_css("this_page.css")
css_merge(css1, css2)

## End(Not run)
```

css_merge_core *A CSS-aware version of nested* modifyList()

Description

If an individual element in the base list has the 'important' attribute set to TRUE, then it does *not* get overwritten.

Usage

```
css_merge_core(base, new)
```

Arguments

base	the original list
new	the new list from which to take values and put into base

Details

Some properties are accumulative, rather than replacement-based. E.g. SVGs transform attribute combines parent + child transform matrices. For now, any accumulative properties are concatenated into a vector of character stirngs to be dealt with by the user after parsing.

Otherwise, this function behaves very much like utils::modifyList()

Value

updated version of base list

css_pretty_print *Pretty Printing of a CSS stylesheet (as produced by* read_css()*)*

Description

Pretty Printing of a CSS stylesheet (as produced by read_css())

Usage

```
css_pretty_print(x, depth = 0, rulesep = "\n", ...)
```

Arguments

x	object representing a CSS stylesheet
depth	the recursion depth of this block (used for indentation)
rulesep	separator between multiple rules
...	other arguments ignored

css_string_as_css_length

Convert a string value into a numeric value with a 'unit' attribute (if a unit is present)

Description

https://www.w3schools.com/CSSref/css_units.asp

Usage

```
css_string_as_css_length(x)
```

Arguments

x	a value from CSS.

Value

css_length object

css_string_as_pixels

Naively convert a CSS string value (e.g. "1em") into the number of pixels this represents.

Description

This function is a small wrapper around `css_string_as_length()` and `css_length_as_pixels()`.

Usage

```
css_string_as_pixels(x, percentage_as_fraction = TRUE, ...)
```

Arguments

x	Character string of a CSS value e.g. "12", "12px", "3em", "47%"
percentage_as_fraction	Default: TRUE means that if a value is given as "50 FALSE, then a numeric value of 50 would be returned.
...	other arguments passed to `css_length_as_pixels()`

Details

This function does some naive conversions, and assumes the display is 96dpi.

For more control on the final result, the user is encouraged to use `css_string_as_length()` and hand-roll their own unit conversion to their display units. This can be tricky as some units rely on rendering viewport sizes and font-size on the root node - thus i'll leave that for the user to handle.

Value

a numeric value for this length in pixels as best we can with limited knowledge

---

`html4_user_agent_css` *HTML4 user-agent css*

Description

HTML4 user-agent css

Usage

```
html4_user_agent_css
```

Format

An object of class `list` of length 120.

`init_stream` *A minimal stream object (reminiscent of an R6 class)*

Description

This tream object (an environment) is used to encapsulate a vector of named tokens, an index of the current position, and some simple tools for advancing the stream, consuming tokens, etc

Usage

```
init_stream(tokens)
```

Arguments

`tokens`	named

is_char1 *Test if string is a single string with something in it*

Description

Test if string is a single string with something in it

Usage

```
is_char1(x)
```

Arguments

x	string

Value

TRUE if string is single string with something in it

lex *Break a string into labelled tokens based upon a set of patterns*

Description

Break a string into labelled tokens based upon a set of patterns

Usage

```
lex(text, regexes, verbose = FALSE)
```

Arguments

text	a single character string
regexes	a named vector of regex strings. Each string represents a regex to match a token, and the name of the string is the label for the token. Each regex can contain an explicit captured group using the standard () brackets. If a regex doesn't not define a captured group then the entire regex will be captured. The regexes will be processed in order such that an early match takes precedence over any later match.
verbose	print more information about the matching process. default: FALSE

Value

a named character vector with the names representing the token type with the value being the element extracted by the corresponding regular expression.

Examples

```
## Not run:
lex("hello there 123.45", regexes=c(number=re$number, word="(\\w+)", whitespace="(\\s+)"))

## End(Not run)
```

`parse_at` *Parse a simple "at' rule from the stream at the current location*

Description

This is really generic and lumps everything up to the next semi-colon into a single character string.

Usage

```
parse_at(stream)
```

Arguments

`stream`	stream

`parse_colour_hex` *Normalize a hex colour to an 8-char hex*

Description

- If hex colour doesn't have an alpha channel, add it as 'FF'
- If hex colour only has 3 digits, expand it to 6, then add alpha channel

Usage

```
parse_colour_hex(hex_colour)
```

Arguments

`hex_colour`	e.g. '#000'

Value

8 char hex colour

Examples

```
## Not run:
parse_colour_hex('#123') # -> '#112233FF'

## End(Not run)
```

parse_colour_hsl_to_hex

Parse a hsl/hsla colour spec to a hex colour

Description

MDN https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/hsl()

Usage

```
parse_colour_hsl_to_hex(colour_func_text)
```

Arguments

colour_func_text	string representing an hsl() or hsla() colour e.g. 'hsl(50, 0.5, 0.5)'

Value

hex colour

Examples

```
## Not run:
parse_colour_hsl_to_hex('hsl(50, 0.5, 0.5)')

## End(Not run)
```

parse_colour_rgb_to_hex

Parse an rgb() or rgba() colour spec to a hex colour

Description

MDN https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/rgb()

Usage

```
parse_colour_rgb_to_hex(rgb_text)
```

Arguments

rgb_text	string representing an rgb() or rgba() colour e.g. 'rgb(255, 0, 0)'

Value

hex colour

Examples

```
## Not run:
parse_colour_rgb_to_hex('rgb(255, 0, 0)')

## End(Not run)
```

`parse_declaration` *Extract a declaration (i.e. a property/value pair) from token stream*

Description

e.g. "fill: black; color: blue" -> list(fill = 'black', color = 'blue')

Usage

```
parse_declaration(stream)
```

Arguments

`stream`	environment containing tokens and the current index. This is an internal only datastructure to assist in parsing

Value

a named list i.e. the name/value property pair

`parse_nested_at` *Parse a 'nested at' rule from the stream at the current location.*

Description

Nested 'ats' are things like `@media` where there is a nested stylesheet as part of this declaration.

Usage

```
parse_nested_at(stream)
```

Arguments

`stream`	stream

Value

name list of list(rule_name = list(property = value, ...))

parse_rule *Parse a rule from the stream at the current location*

Description

A rule here is a selector and a list of property/value pairs

Usage

```
parse_rule(stream)
```

Arguments

stream	stream

Value

name list of list(rule_name = list(property = value, ...))

parse_rules_from_stream *Parse rules from the stream at the current location*

Description

A 'rule' is a 'selector' + a list of property/value pairs.

Usage

```
parse_rules_from_stream(stream)
```

Arguments

stream	token stream

Value

name list of list(rule_name = list(property = value, ...))

parse_selectors *Extract selector from CSS tokens stream*

Description

The selector is everything from the start of the rule to the first open-brace

Usage

```
parse_selectors(stream)
```

Arguments

stream	environment containing tokens and the current index. This is an internal only datastructure to assist in parsing

Details

e.g. .circle fill = 'black' -> '.circle'

Value

single character string

print.stream *s3 method fr printing a stream*

Description

s3 method fr printing a stream

Usage

```
## S3 method for class 'stream'
print(x, ...)
```

Arguments

x	stream object
...	other arguments passed to print()

read_css — *Convert CSS stylesheet from text or a file into a list object*

Description

Convert CSS stylesheet from text or a file into a list object

Usage

```
read_css(stylesheet)
```

Arguments

stylesheet	CSS stylsheet. Either as a character string containing the CSS, or a path to a file containing CSS text

Value

named list of lists, where the top-level name is the CSS selector, and the value is a list of property/value pairs for this selector (i.e. a CSS declaration block)

read_inline_style — *Parse CSS declaration block (aka inline style) to a named list of CSS declarations*

Description

Parse CSS declaration block (aka inline style) to a named list of CSS declarations

Usage

```
read_inline_style(inline_style)
```

Arguments

inline_style	set of ";"-delmited declarations i.e. an inline style, or a CSS declaration block: e.g. "stroke:red; fill: black; stroke-width:12"

Value

named list of declarations (i.e. property/value pairs) e.g. `list(stroke = 'red', fill = 'black', 'stroke-width' = 12)`

recurse_node_tree *Depth-first recursion into the HTML document node tree to accumulate/cascade styles*

Description

Depth-first recursion into the HTML document node tree to accumulate/cascade styles

Usage

```
recurse_node_tree(node, xpath_styles, parent_style = list(), svg)
```

Arguments

node	current XML node
xpath_styles	list containing the accumulated styles for each xpath
parent_style	the style from the direct parent
svg	include SVG presentation tags in the cascade? default: FALSE

Value

updated xpath style lookup list

selector_specificity *Calculate selector specificity*

Description

Algorithm defined here: https://drafts.csswg.org/selectors-3/#specificity

Usage

```
selector_specificity(sel)
```

Arguments

sel	selector

Details

A selector's specificity is calculated as follows:

- count the number of ID selectors in the selector (= a)
- count the number of class selectors, attributes selectors, and pseudo-classes in the selector (= b)
- count the number of type selectors and pseudo-elements in the selector (= c)
- ignore the universal selector

Selectors inside the negation pseudo-class are counted like any other, but the negation itself does not count as a pseudo-class.

Concatenating the three numbers a-b-c (in a number system with a large base) gives the specificity.

Since we only really have base-10 numbers (rather than numbers in arbitrarily large bases), just use 2 digits for each of the counts. Hopefully it is really unlikely that any of the counts exceeds 99!

Value

numeric value

---

split_font_shorthand *Split font shorthand property into a named list of verbose "font-*" properties*

Description

This is not perfect/finished but it is a step in the right direction.

Usage

```
split_font_shorthand(font_text)
```

Arguments

font_text	the text of the "font" declaration e.g. "italic 1.2em "Fira Sans", serif;"

Details

For example: this function does not split the "font-variant" shorthand

Value

list of font-related property/value declarations e.g. `list('font-family'='serif', font-size='1.2em', ...)`

style_flatten_to_inline *Flatten a style to an inline string*

Description

Flatten a style to an inline string

Usage

```
style_flatten_to_inline(style)
```

Arguments

style	a named list of property/value pairs

Value

single string suitable for a `style` attribute on an element

Examples

```
## Not run:
style_flatten_to_line(list(color='black', border='1px')) # -> "color:black; border: 1px;"

## End(Not run)
```

style_merge *Merge multiple styles for a given element which are given in priority order*

Description

Merge multiple styles for a given element which are given in priority order

Usage

```
style_merge(...)
```

Arguments

...	multiple style objects (i.e. declaration blocks). The order in which these arguments are given reflect the priority of the styles from lowest to highest priority. Later styles (i.e. high priority) will override any styles declared earlier (lower priority)

Value

final cascaded result

Examples

```
## Not run:
style1 <- list(color = 'blue', `font-weight` = 'bold')
style2 <- list(color = 'red', `font-size`="12px")
style_merge(style1, style2)

## End(Not run)
```

---

style_pretty_print *Pretty Printing of a single style or declaration block*

Description

Pretty Printing of a single style or declaration block

Usage

```
style_pretty_print(x, depth = 0)
```

Arguments

x	object representing a list of property/value declarations
depth	the recursion depth of this block (used for indentation)

---

trim *Trim the end off a string*

Description

Trim the end off a string

Usage

```
trim(x, len)
```

Arguments

x	string
len	number of characters to trim

`xml_duplicate` *Make a distinct, unlinked, independent copy of an xml node*

Description

Since XML documents and nodes are almost always handled 'by-reference' if you want to keep an original untouched when adding nodes etc, you need to make an unlinked copy of it.

Usage

```
xml_duplicate(x)
```

Arguments

`x`	xml document

Value

new, unlinked xml document

Index