Package ‘js’

July 28, 2017

Type Package
Title Tools for Working with JavaScript in R
Version 1.1
Author Jeroen Ooms
Maintainer Jeroen Ooms <jeroen@berkeley.edu>
Description A set of utilities for working with JavaScript syntax in R.
   Includes tools to parse, tokenize, compile, validate, reformat, optimize
   and analyze JavaScript code.
License MIT + file LICENSE
URL https://github.com/jeroen/js
BugReports https://github.com/jeroen/js/issues
VignetteBuilder knitr
Imports V8 (>= 0.5)
Suggests knitr, rmarkdown
RoxygenNote 6.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2017-07-28 21:32:04 UTC

R topics documented:

coffee_compile .................................................. 2
esprima .......................................................... 2
jshint ........................................................... 3
js_eval ............................................................ 4
js_typeof ........................................................... 4
js_validate_script .............................................. 5
uglify ............................................................. 5

Index 7
coffee_compile  Coffee Script

Description

Compiles coffee script into JavaScript.

Usage

coffee_compile(code, ...)

Arguments

code  a string with JavaScript code

... additional options passed to the compiler

Examples

# Hello world
coffee_compile("square = (x) -> x * x")
coffee_compile("square = (x) -> x * x", bare = TRUE)

# Simple script
demo <- readLines(system.file("example/demo.coffee", package = "js"))
js <- coffee_compile(demo)
cat(js)
cat(uglify_optimize(js))

esprima  JavaScript Syntax Tree

Description

Esprima is a high performance, standard-compliant ECMAScript parser. It has full support for ECMAScript 2017 and returns a sensible syntax tree format as standardized by ESTree project.

Usage

esprima_tokenize(text, range = FALSE, loc = FALSE, comment = FALSE)

esprima_parse(text, jsx = FALSE, range = FALSE, loc = FALSE,

tolerant = FALSE, tokens = FALSE, comment = FALSE)
Arguments

- **text**: a character vector with JavaScript code
- **range**: Annotate each token with its zero-based start and end location
- **loc**: Annotate each token with its column and row-based location
- **comment**: Include every line and block comment in the output
- **jsx**: Support JSX syntax
- **tolerant**: Tolerate a few cases of syntax errors
- **tokens**: Collect every token

Details

The `esprima_tokenize` function returns a data frame with JavaScript tokens. The `esprima_parse` function returns the Syntax Tree in JSON format. This can be parsed to R using e.g. `jsonlite::fromJSON`.

References


Examples

```r
code <- "function test(x, y){ x = x || 1; y = y || 1; return x*y;}

esprima_tokenize(code)
esprima_parse(code)
```

---

**jshint**

*Static analysis tool for JavaScript*

Description

JSHint is a community-driven tool to detect errors and potential problems in JavaScript code. It is very flexible so you can easily adjust it to your particular coding guidelines and the environment you expect your code to execute in.

Usage

```r
jshint(text, ..., globals = NULL)
```

Arguments

- **text**: a string of JavaScript code
- **...**: additional jshint configuration options
- **globals**: a white list of global variables that are not formally defined in the source code

Value

A data frame where each row represents a jshint error or `NULL` if there were no errors.
Examples

```javascript
code = "var foo = 123"
jshint(code)
jshint(code, asi = TRUE)
```

---

**js_eval**  
*Evaluate JavaScript*

**Description**

Evaluate a piece of JavaScript code in a disposable context.

**Usage**

```javascript
js_eval(text)
```

**Arguments**

- text  
  JavaScript code

**Examples**

```javascript
# Stateless evaluation
js_eval("(function() {return 'foo'}]()")

# Use V8 for stateful evaluation
ct <- v8::new_context()
ct$eval("var foo = 123")
ct$get("foo")
```

---

**js_typeof**  
*Get the type of a JavaScript object*

**Description**

JavaScript wrapper to `typeof` to test if a piece of JavaScript code is syntactically valid, and the type of object it evaluates to. Useful to verify that a piece of JavaScript code contains a proper function/object.

**Usage**

```javascript
js_typeof(text)
```

**Arguments**

- text  
  JavaScript code
js_validate_script

Examples

- `js_typeof("function(x){return x+1}")`
- `js_typeof("function() {return 'foo'}()")`
- `js_typeof("{foo : 123, bar : true}")`

Description

Simple wrapper for ct-validate in V8. Tests if code constitutes a syntactically valid JS script.

Usage

- `js_validate_script(text, error = TRUE)`

Arguments

- `text`: character vector with JavaScript code
- `error`: raise error on invalid code

Examples

- `js_validate_script("function foo(x){(x)}") #TRUE`
- `js_validate_script("foo = function(x){(x)}") #TRUE`

# Anonymous functions in global scope are invalid
- `js_validate_script("function(x){(x)}", error = FALSE) #FALSE`

# Use ! or () to check anonymous function syntax
- `js_validate_script("!function(x){(x)}") #TRUE`
- `js_validate_script("(function(x){(x)})") #TRUE`

uglify

Compress and Reformat JavaScript Code

Description

UglifyJS is a JavaScript compressor/minifier written in JavaScript. It also contains tools that allow one to automate working with JavaScript code.

Usage

- `uglify_reformat(text, beautify = FALSE, ...)`
- `uglify_optimize(text, ...)"
Arguments

- **text**: a character vector with JavaScript code
- **beautify**: prettify (instead of minify) code
- **...**: additional arguments for the optimizer or generator.

References

UglifyJS2 Documentation: [http://lisperator.net/uglifyjs/](http://lisperator.net/uglifyjs/).

Examples

code <- "function test(x, y){ x = x || 1; y = y || 1; return x*y;}"
cat(uglify_optimize(code))
cat(uglify_reformat(code, beautify = TRUE, indent_level = 2))
Index

coffee (coffee_compile), 2
coffee_compile, 2

esprima, 2
esprima_parse (esprima), 2
esprima_tokenize (esprima), 2

js_eval, 4
js_typeof, 4
js_validate_script, 5
jshint, 3

uglify, 5
uglify_optimize (uglify), 5
uglify_reformat (uglify), 5

V8, 5