

[![NPM version](https://img.shields.io/npm/v/clean-css.svg?style=flat)](https://www.npmjs.com/package/clean-css) [![Linux Build Status](https://img.shields.io/travis/jakubpawlowicz/clean-css/master.svg?style=flat&label=Linux%20build)](https://travis-ci.org/jakubpawlowicz/clean-css) [![Windows Build status](https://img.shields.io/appveyor/ci/jakubpawlowicz/clean-css/master.svg?style=flat&label=Windows%20build)](https://ci.appveyor.com/project/jakubpawlowicz/clean-css/branch/master) [![Dependency Status](https://img.shields.io/david/jakubpawlowicz/clean-css.svg?style=flat)](https://david-dm.org/jakubpawlowicz/clean-css) [![NPM Downloads](https://img.shields.io/npm/dm/clean-css.svg)](https://npmcharts.com/compare/clean-css?minimal=true) [![Twitter](https://img.shields.io/badge/Twitter-@cleancss-blue.svg)](https://twitter.com/cleancss) clean-css is a fast and efficient CSS optimizer for [Node.js](http://nodejs.org/) platform and [any modern browser](https://jakubpawlowicz.github.io/clean-css). According to [tests](http://goalsmashers.github.io/css-minification-benchmark/) it is one of the best available. \*\*Table of Contents\*\* - [Node.js version support](#nodejs-version-support) - [Install](#install) - [Use](#use) \* [Important: 4.0 breaking changes](#important-40-breaking-changes) \* [What's new in version 4.1](#whats-new-in-version-41) \* [What's new in version 4.2](#whats-new-in-version-42) \* [Constructor options](#constructor-options) \* [Compatibility modes](#compatibility-modes) \* [Fetch option](#fetch-option) \* [Formatting options](#formatting-options) \* [Inlining options](#inlining-options) \* [Optimization levels](#optimization-levels) + [Level 0 optimizations](#level-0-optimizations) + [Level 1 optimizations](#level-1-optimizations) + [Level 2 optimizations](#level-2-optimizations) \* [Minify method](#minify-method) \* [Promise interface](#promise-interface) \* [CLI utility](#cli-utility) - [FAQ](#faq) \* [How to optimize multiple files?](#how-to-optimize-multiple-files) \* [How to process remote `@import`s correctly?](#how-to-process-remote-imports-correctly) \* [How to apply arbitrary transformations to CSS properties?](#how-to-apply-arbitrary-transformations-to-css-properties) \* [How to specify a custom rounding precision?](#how-to-specify-a-custom-rounding-precision) \* [How to keep a CSS fragment intact?](#how-to-keep-a-css-fragment-intact) \* [How to preserve a comment block?](#how-to-preserve-a-comment-block) \* [How to rebase relative image URLs?](#how-to-rebase-relative-image-urls) \* [How to work with source maps?](#how-to-work-with-source-maps) \* [How to apply level 1 & 2 optimizations at the same time?](#how-to-apply-level-1--2-optimizations-at-the-same-time) \* [What level 2 optimizations do?](#what-level-2-optimizations-do) \* [How to use clean-css with build tools?](#how-to-use-clean-css-with-build-tools) \* [How to use clean-css from web browser?](#how-to-use-clean-css-from-web-browser) - [Contributing](#contributing) \* [How to get started?](#how-to-get-started) - [Acknowledgments](#acknowledgments) - [License](#license) # Node.js version support clean-css requires Node.js 4.0+ (tested on Linux, OS X, and Windows) # Install ``` npm install --save-dev clean-css ``` # Use ```js var CleanCSS = require('clean-css'); var input = 'a{font-weight:bold;}'; var options = { /\* options \*/ }; var output = new CleanCSS(options).minify(input); ``` ## Important: 4.0 breaking changes clean-css 4.0 introduces some breaking changes: \* API and CLI interfaces are split, so API stays in this repository while CLI moves to [clean-css-cli](https://github.com/jakubpawlowicz/clean-css-cli); \* `root`, `relativeTo`, and `target` options are replaced by a single `rebaseTo` option - this means that rebasing URLs and import inlining is much simpler but may not be (YMMV) as powerful as in 3.x; \* `debug` option is gone as stats are always provided in output object under `stats` property; \* `roundingPrecision` is disabled by default; \* `roundingPrecision` applies to \*\*all\*\* units now, not only `px` as in 3.x; \* `processImport` and `processImportFrom` are merged into `inline` option which defaults to `local`. Remote `@import` rules are \*\*NOT\*\* inlined by default anymore; \* splits `inliner: { request: ..., timeout: ... }` option into `inlineRequest` and `inlineTimeout` options; \* remote resources without a protocol, e.g. `//fonts.googleapis.com/css?family=Domine:700`, are not inlined anymore; \* changes default Internet Explorer compatibility from 9+ to 10+, to revert the old default use `{ compatibility: 'ie9' }` flag; \* renames `keepSpecialComments` to `specialComments`; \* moves `roundingPrecision` and `specialComments` to level 1 optimizations options, see examples; \* moves `mediaMerging`, `restructuring`, `semanticMerging`, and `shorthandCompacting` to level 2 optimizations options, see examples below; \* renames `shorthandCompacting` option to `mergeIntoShorthands`; \* level 1 optimizations are the new default, up to 3.x it was level 2; \* `keepBreaks` option is replaced with `{ format: 'keep-breaks' }` to ease transition; \* `sourceMap` option has to be a boolean from now on - to specify an input source map pass it a 2nd argument to `minify` method or via a hash instead; \* `aggressiveMerging` option is removed as aggressive merging is replaced by smarter override merging. ## What's new in version 4.1 clean-css 4.1 introduces the following changes / features: \* `inline: false` as an alias to `inline: ['none']`; \* `multiplePseudoMerging` compatibility flag controlling merging of rules with multiple pseudo classes / elements; \* `removeEmpty` flag in level 1 optimizations controlling removal of rules and nested blocks; \* `removeEmpty` flag in level 2 optimizations controlling removal of rules and nested blocks; \* `compatibility: { selectors: { mergeLimit: } }` flag in compatibility settings controlling maximum number of selectors in a single rule; \* `minify` method improved signature accepting a list of hashes for a predictable traversal; \* `selectorsSortingMethod` level 1 optimization allows `false` or `'none'` for disabling selector sorting; \* `fetch` option controlling a function for handling remote requests; \* new `font` shorthand and `font-\*` longhand optimizers; \* removal of `optimizeFont` flag in level 1 optimizations due to new `font` shorthand optimizer; \* `skipProperties` flag in level 2 optimizations controlling which properties won't be optimized; \* new `animation` shorthand and `animation-\*` longhand optimizers; \* `removeUnusedAtRules` level 2 optimization controlling removal of unused `@counter-style`, `@font-face`, `@keyframes`, and `@namespace` at rules; \* the [web interface](https://jakubpawlowicz.github.io/clean-css) gets an improved settings panel with "reset to defaults", instant option changes, and settings being persisted across sessions. ## What's new in version 4.2 clean-css 4.2 introduces the following changes / features: \* Adds `process` method for compatibility with optimize-css-assets-webpack-plugin; \* new `transition` property optimizer; \* preserves any CSS content between `/\* clean-css ignore:start \*/` and `/\* clean-css ignore:end \*/` comments; \* allows filtering based on selector in `transform` callback, see [example](#how-to-apply-arbitrary-transformations-to-css-properties); \* adds configurable line breaks via `format: { breakWith: 'lf' }` option. ## Constructor options clean-css constructor accepts a hash as a parameter with the following options available: \* `compatibility` - controls compatibility mode used; defaults to `ie10+`; see [compatibility modes](#compatibility-modes) for examples; \* `fetch` - controls a function for handling remote requests; see [fetch option](#fetch-option) for examples (since 4.1.0); \* `format` - controls output CSS formatting; defaults to `false`; see [formatting options](#formatting-options) for examples; \* `inline` - controls `@import` inlining rules; defaults to `'local'`; see [inlining options](#inlining-options) for examples; \* `inlineRequest` - controls extra options for inlining remote `@import` rules, can be any of [HTTP(S) request options](https://nodejs.org/api/http.html#http\_http\_request\_options\_callback); \* `inlineTimeout` - controls number of milliseconds after which inlining a remote `@import` fails; defaults to 5000; \* `level` - controls optimization level used; defaults to `1`; see [optimization levels](#optimization-levels) for examples; \* `rebase` - controls URL rebasing; defaults to `true`; \* `rebaseTo` - controls a directory to which all URLs are rebased, most likely the directory under which the output file will live; defaults to the current directory; \* `returnPromise` - controls whether `minify` method returns a Promise object or not; defaults to `false`; see [promise interface](#promise-interface) for examples; \* `sourceMap` - controls whether an output source map is built; defaults to `false`; \* `sourceMapInlineSources` - controls embedding sources inside a source map's `sourcesContent` field; defaults to false. ## Compatibility modes There is a certain number of compatibility mode shortcuts, namely: \* `new CleanCSS({ compatibility: '\*' })` (default) - Internet Explorer 10+ compatibility mode \* `new CleanCSS({ compatibility: 'ie9' })` - Internet Explorer 9+ compatibility mode \* `new CleanCSS({ compatibility: 'ie8' })` - Internet Explorer 8+ compatibility mode \* `new CleanCSS({ compatibility: 'ie7' })` - Internet Explorer 7+ compatibility mode Each of these modes is an alias to a [fine grained configuration](https://github.com/jakubpawlowicz/clean-css/blob/master/lib/options/compatibility.js), with the following options available: ```js new CleanCSS({ compatibility: { colors: { opacity: true // controls `rgba()` / `hsla()` color support }, properties: { backgroundClipMerging: true, // controls background-clip merging into shorthand backgroundOriginMerging: true, // controls background-origin merging into shorthand backgroundSizeMerging: true, // controls background-size merging into shorthand colors: true, // controls color optimizations ieBangHack: false, // controls keeping IE bang hack ieFilters: false, // controls keeping IE `filter` / `-ms-filter` iePrefixHack: false, // controls keeping IE prefix hack ieSuffixHack: false, // controls keeping IE suffix hack merging: true, // controls property merging based on understandability shorterLengthUnits: false, // controls shortening pixel units into `pc`, `pt`, or `in` units spaceAfterClosingBrace: true, // controls keeping space after closing brace - `url() no-repeat` into `url()no-repeat` urlQuotes: false, // controls keeping quoting inside `url()` zeroUnits: true // controls removal of units `0` value }, selectors: { adjacentSpace: false, // controls extra space before `nav` element ie7Hack: true, // controls removal of IE7 selector hacks, e.g. `\*+html...` mergeablePseudoClasses: [':active', ...], // controls a whitelist of mergeable pseudo classes mergeablePseudoElements: ['::after', ...], // controls a whitelist of mergeable pseudo elements mergeLimit: 8191, // controls maximum number of selectors in a single rule (since 4.1.0) multiplePseudoMerging: true // controls merging of rules with multiple pseudo classes / elements (since 4.1.0) }, units: { ch: true, // controls treating `ch` as a supported unit in: true, // controls treating `in` as a supported unit pc: true, // controls treating `pc` as a supported unit pt: true, // controls treating `pt` as a supported unit rem: true, // controls treating `rem` as a supported unit vh: true, // controls treating `vh` as a supported unit vm: true, // controls treating `vm` as a supported unit vmax: true, // controls treating `vmax` as a supported unit vmin: true // controls treating `vmin` as a supported unit } } }) ``` You can also use a string when setting a compatibility mode, e.g. ```js new CleanCSS({ compatibility: 'ie9,-properties.merging' // sets compatibility to IE9 mode with disabled property merging }) ``` ## Fetch option The `fetch` option accepts a function which handles remote resource fetching, e.g. ```js var request = require('request'); var source = '@import url(http://example.com/path/to/stylesheet.css);'; new CleanCSS({ fetch: function (uri, inlineRequest, inlineTimeout, callback) { request(uri, function (error, response, body) { if (error) { callback(error, null); } else if (response && response.statusCode != 200) { callback(response.statusCode, null); } else { callback(null, body); } }); } }).minify(source); ``` This option provides a convenient way of overriding the default fetching logic if it doesn't support a particular feature, say CONNECT proxies. Unless given, the default [loadRemoteResource](https://github.com/jakubpawlowicz/clean-css/blob/master/lib/reader/load-remote-resource.js) logic is used. ## Formatting options By default output CSS is formatted without any whitespace unless a `format` option is given. First of all there are two shorthands: ```js new CleanCSS({ format: 'beautify' // formats output in a really nice way }) ``` and ```js new CleanCSS({ format: 'keep-breaks' // formats output the default way but adds line breaks for improved readability }) ``` however `format` option also accept a fine-grained set of options: ```js new CleanCSS({ format: { breaks: { // controls where to insert breaks afterAtRule: false, // controls if a line break comes after an at-rule; e.g. `@charset`; defaults to `false` afterBlockBegins: false, // controls if a line break comes after a block begins; e.g. `@media`; defaults to `false` afterBlockEnds: false, // controls if a line break comes after a block ends, defaults to `false` afterComment: false, // controls if a line break comes after a comment; defaults to `false` afterProperty: false, // controls if a line break comes after a property; defaults to `false` afterRuleBegins: false, // controls if a line break comes after a rule begins; defaults to `false` afterRuleEnds: false, // controls if a line break comes after a rule ends; defaults to `false` beforeBlockEnds: false, // controls if a line break comes before a block ends; defaults to `false` betweenSelectors: false // controls if a line break comes between selectors; defaults to `false` }, breakWith: '\n', // controls the new line character, can be `'\r\n'` or `'\n'` (aliased as `'windows'` and `'unix'` or `'crlf'` and `'lf'`); defaults to system one, so former on Windows and latter on Unix indentBy: 0, // controls number of characters to indent with; defaults to `0` indentWith: 'space', // controls a character to indent with, can be `'space'` or `'tab'`; defaults to `'space'` spaces: { // controls where to insert spaces aroundSelectorRelation: false, // controls if spaces come around selector relations; e.g. `div > a`; defaults to `false` beforeBlockBegins: false, // controls if a space comes before a block begins; e.g. `.block {`; defaults to `false` beforeValue: false // controls if a space comes before a value; e.g. `width: 1rem`; defaults to `false` }, wrapAt: false // controls maximum line length; defaults to `false` } }) ``` ## Inlining options `inline` option whitelists which `@import` rules will be processed, e.g. ```js new CleanCSS({ inline: ['local'] // default; enables local inlining only }) ``` ```js new CleanCSS({ inline: ['none'] // disables all inlining }) ``` ```js // introduced in clean-css 4.1.0 new CleanCSS({ inline: false // disables all inlining (alias to `['none']`) }) ``` ```js new CleanCSS({ inline: ['all'] // enables all inlining, same as ['local', 'remote'] }) ``` ```js new CleanCSS({ inline: ['local', 'mydomain.example.com'] // enables local inlining plus given remote source }) ``` ```js new CleanCSS({ inline: ['local', 'remote', '!fonts.googleapis.com'] // enables all inlining but from given remote source }) ``` ## Optimization levels The `level` option can be either `0`, `1` (default), or `2`, e.g. ```js new CleanCSS({ level: 2 }) ``` or a fine-grained configuration given via a hash. Please note that level 1 optimization options are generally safe while level 2 optimizations should be safe for most users. ### Level 0 optimizations Level 0 optimizations simply means "no optimizations". Use it when you'd like to inline imports and / or rebase URLs but skip everything else. ### Level 1 optimizations Level 1 optimizations (default) operate on single properties only, e.g. can remove units when not required, turn rgb colors to a shorter hex representation, remove comments, etc Here is a full list of available options: ```js new CleanCSS({ level: { 1: { cleanupCharsets: true, // controls `@charset` moving to the front of a stylesheet; defaults to `true` normalizeUrls: true, // controls URL normalization; defaults to `true` optimizeBackground: true, // controls `background` property optimizations; defaults to `true` optimizeBorderRadius: true, // controls `border-radius` property optimizations; defaults to `true` optimizeFilter: true, // controls `filter` property optimizations; defaults to `true` optimizeFont: true, // controls `font` property optimizations; defaults to `true` optimizeFontWeight: true, // controls `font-weight` property optimizations; defaults to `true` optimizeOutline: true, // controls `outline` property optimizations; defaults to `true` removeEmpty: true, // controls removing empty rules and nested blocks; defaults to `true` removeNegativePaddings: true, // controls removing negative paddings; defaults to `true` removeQuotes: true, // controls removing quotes when unnecessary; defaults to `true` removeWhitespace: true, // controls removing unused whitespace; defaults to `true` replaceMultipleZeros: true, // contols removing redundant zeros; defaults to `true` replaceTimeUnits: true, // controls replacing time units with shorter values; defaults to `true` replaceZeroUnits: true, // controls replacing zero values with units; defaults to `true` roundingPrecision: false, // rounds pixel values to `N` decimal places; `false` disables rounding; defaults to `false` selectorsSortingMethod: 'standard', // denotes selector sorting method; can be `'natural'` or `'standard'`, `'none'`, or false (the last two since 4.1.0); defaults to `'standard'` specialComments: 'all', // denotes a number of /\*! ... \*/ comments preserved; defaults to `all` tidyAtRules: true, // controls at-rules (e.g. `@charset`, `@import`) optimizing; defaults to `true` tidyBlockScopes: true, // controls block scopes (e.g. `@media`) optimizing; defaults to `true` tidySelectors: true, // controls selectors optimizing; defaults to `true`, semicolonAfterLastProperty: false, // controls removing trailing semicolons in rule; defaults to `false` - means remove transform: function () {} // defines a callback for fine-grained property optimization; defaults to no-op } } }); ``` There is an `all` shortcut for toggling all options at the same time, e.g. ```js new CleanCSS({ level: { 1: { all: false, // set all values to `false` tidySelectors: true // turns on optimizing selectors } } }); ``` ### Level 2 optimizations Level 2 optimizations operate at rules or multiple properties level, e.g. can remove duplicate rules, remove properties redefined further down a stylesheet, or restructure rules by moving them around. Please note that if level 2 optimizations are turned on then, unless explicitely disabled, level 1 optimizations are applied as well. Here is a full list of available options: ```js new CleanCSS({ level: { 2: { mergeAdjacentRules: true, // controls adjacent rules merging; defaults to true mergeIntoShorthands: true, // controls merging properties into shorthands; defaults to true mergeMedia: true, // controls `@media` merging; defaults to true mergeNonAdjacentRules: true, // controls non-adjacent rule merging; defaults to true mergeSemantically: false, // controls semantic merging; defaults to false overrideProperties: true, // controls property overriding based on understandability; defaults to true removeEmpty: true, // controls removing empty rules and nested blocks; defaults to `true` reduceNonAdjacentRules: true, // controls non-adjacent rule reducing; defaults to true removeDuplicateFontRules: true, // controls duplicate `@font-face` removing; defaults to true removeDuplicateMediaBlocks: true, // controls duplicate `@media` removing; defaults to true removeDuplicateRules: true, // controls duplicate rules removing; defaults to true removeUnusedAtRules: false, // controls unused at rule removing; defaults to false (available since 4.1.0) restructureRules: false, // controls rule restructuring; defaults to false skipProperties: [] // controls which properties won't be optimized, defaults to `[]` which means all will be optimized (since 4.1.0) } } }); ``` There is an `all` shortcut for toggling all options at the same time, e.g. ```js new CleanCSS({ level: { 2: { all: false, // sets all values to `false` removeDuplicateRules: true // turns on removing duplicate rules } } }); ``` ## Minify method Once configured clean-css provides a `minify` method to optimize a given CSS, e.g. ```js var output = new CleanCSS(options).minify(source); ``` The output of the `minify` method is a hash with following fields: ```js console.log(output.styles); // optimized output CSS as a string console.log(output.sourceMap); // output source map if requested with `sourceMap` option console.log(output.errors); // a list of errors raised console.log(output.warnings); // a list of warnings raised console.log(output.stats.originalSize); // original content size after import inlining console.log(output.stats.minifiedSize); // optimized content size console.log(output.stats.timeSpent); // time spent on optimizations in milliseconds console.log(output.stats.efficiency); // `(originalSize - minifiedSize) / originalSize`, e.g. 0.25 if size is reduced from 100 bytes to 75 bytes ``` The `minify` method also accepts an input source map, e.g. ```js var output = new CleanCSS(options).minify(source, inputSourceMap); ``` or a callback invoked when optimizations are finished, e.g. ```js new CleanCSS(options).minify(source, function (error, output) { // `output` is the same as in the synchronous call above }); ``` ## Promise interface If you prefer clean-css to return a Promise object then you need to explicitely ask for it, e.g. ```js new CleanCSS({ returnPromise: true }) .minify(source) .then(function (output) { console.log(output.styles); }) .catch(function (error) { // deal with errors }); ``` ## CLI utility Clean-css has an associated command line utility that can be installed separately using `npm install clean-css-cli`. For more detailed information, please visit https://github.com/jakubpawlowicz/clean-css-cli. # FAQ ## How to optimize multiple files? It can be done either by passing an array of paths, or, when sources are already available, a hash or an array of hashes: ```js new CleanCSS().minify(['path/to/file/one', 'path/to/file/two']); ``` ```js new CleanCSS().minify({ 'path/to/file/one': { styles: 'contents of file one' }, 'path/to/file/two': { styles: 'contents of file two' } }); ``` ```js new CleanCSS().minify([ {'path/to/file/one': {styles: 'contents of file one'}}, {'path/to/file/two': {styles: 'contents of file two'}} ]); ``` Passing an array of hashes allows you to explicitly specify the order in which the input files are concatenated. Whereas when you use a single hash the order is determined by the [traversal order of object properties](http://2ality.com/2015/10/property-traversal-order-es6.html) - available since 4.1.0. Important note - any `@import` rules already present in the hash will be resolved in memory. ## How to process remote `@import`s correctly? In order to inline remote `@import` statements you need to provide a callback to minify method as fetching remote assets is an asynchronous operation, e.g.: ```js var source = '@import url(http://example.com/path/to/remote/styles);'; new CleanCSS({ inline: ['remote'] }).minify(source, function (error, output) { // output.styles }); ``` If you don't provide a callback, then remote `@import`s will be left as is. ## How to apply arbitrary transformations to CSS properties? If clean-css doesn't perform a particular property optimization, you can use `transform` callback to apply it: ```js var source = '.block{background-image:url(/path/to/image.png)}'; var output = new CleanCSS({ level: { 1: { transform: function (propertyName, propertyValue, selector /\* `selector` available since 4.2.0-pre \*/) { if (propertyName == 'background-image' && propertyValue.indexOf('/path/to') > -1) { return propertyValue.replace('/path/to', '../valid/path/to'); } } } } }).minify(source); console.log(output.styles); # => .block{background-image:url(../valid/path/to/image.png)} ``` Note: returning `false` from `transform` callback will drop a property. ## How to specify a custom rounding precision? The level 1 `roundingPrecision` optimization option accept a string with per-unit rounding precision settings, e.g. ```js new CleanCSS({ level: { 1: { roundingPrecision: 'all=3,px=5' } } }).minify(source) ``` which sets all units rounding precision to 3 digits except `px` unit precision of 5 digits. ## How to keep a CSS fragment intact? Note: available in the current master, to be released in 4.2.0. Wrap the CSS fragment in special comments which instruct clean-css to preserve it, e.g. ```css .block-1 { color: red } /\* clean-css ignore:start \*/ .block-special { color: transparent } /\* clean-css ignore:end \*/ .block-2 { margin: 0 } ``` Optimizing this CSS will result in the following output: ```css .block-1{color:red} .block-special { color: transparent } .block-2{margin:0} ``` ## How to preserve a comment block? Use the `/\*!` notation instead of the standard one `/\*`: ```css /\*! Important comments included in optimized output. \*/ ``` ## How to rebase relative image URLs? clean-css will handle it automatically for you in the following cases: \* when full paths to input files are passed in as options; \* when correct paths are passed in via a hash; \* when `rebaseTo` is used with any of above two. ## How to work with source maps? To generate a source map, use `sourceMap: true` option, e.g.: ```js new CleanCSS({ sourceMap: true, rebaseTo: pathToOutputDirectory }) .minify(source, function (error, output) { // access output.sourceMap for SourceMapGenerator object // see https://github.com/mozilla/source-map/#sourcemapgenerator for more details }); ``` You can also pass an input source map directly as a 2nd argument to `minify` method: ```js new CleanCSS({ sourceMap: true, rebaseTo: pathToOutputDirectory }) .minify(source, inputSourceMap, function (error, output) { // access output.sourceMap to access SourceMapGenerator object // see https://github.com/mozilla/source-map/#sourcemapgenerator for more details }); ``` or even multiple input source maps at once: ```js new CleanCSS({ sourceMap: true, rebaseTo: pathToOutputDirectory }).minify({ 'path/to/source/1': { styles: '...styles...', sourceMap: '...source-map...' }, 'path/to/source/2': { styles: '...styles...', sourceMap: '...source-map...' } }, function (error, output) { // access output.sourceMap as above }); ``` ## How to apply level 1 & 2 optimizations at the same time? Using the hash configuration specifying both optimization levels, e.g. ```js new CleanCSS({ level: { 1: { all: true, normalizeUrls: false }, 2: { restructureRules: true } } }) ``` will apply level 1 optimizations, except url normalization, and default level 2 optimizations with rule restructuring. ## What level 2 optimizations do? All level 2 optimizations are dispatched [here](https://github.com/jakubpawlowicz/clean-css/blob/master/lib/optimizer/level-2/optimize.js#L67), and this is what they do: \* `recursivelyOptimizeBlocks` - does all the following operations on a nested block, like `@media` or `@keyframe`; \* `recursivelyOptimizeProperties` - optimizes properties in rulesets and flat at-rules, like @font-face, by splitting them into components (e.g. `margin` into `margin-(bottom|left|right|top)`), optimizing, and restoring them back. You may want to use `mergeIntoShorthands` option to control whether you want to turn multiple components into shorthands; \* `removeDuplicates` - gets rid of duplicate rulesets with exactly the same set of properties, e.g. when including a Sass / Less partial twice for no good reason; \* `mergeAdjacent` - merges adjacent rulesets with the same selector or rules; \* `reduceNonAdjacent` - identifies which properties are overridden in same-selector non-adjacent rulesets, and removes them; \* `mergeNonAdjacentBySelector` - identifies same-selector non-adjacent rulesets which can be moved (!) to be merged, requires all intermediate rulesets to not redefine the moved properties, or if redefined to have the same value; \* `mergeNonAdjacentByBody` - same as the one above but for same-selector non-adjacent rulesets; \* `restructure` - tries to reorganize different-selector different-rules rulesets so they take less space, e.g. `.one{padding:0}.two{margin:0}.one{margin-bottom:3px}` into `.two{margin:0}.one{padding:0;margin-bottom:3px}`; \* `removeDuplicateFontAtRules` - removes duplicated `@font-face` rules; \* `removeDuplicateMediaQueries` - removes duplicated `@media` nested blocks; \* `mergeMediaQueries` - merges non-adjacent `@media` at-rules by the same rules as `mergeNonAdjacentBy\*` above; ## How to use clean-css with build tools? There is a number of 3rd party plugins to popular build tools: \* [Broccoli](https://github.com/broccolijs/broccoli#broccoli): [broccoli-clean-css](https://github.com/shinnn/broccoli-clean-css) \* [Brunch](http://brunch.io/): [clean-css-brunch](https://github.com/brunch/clean-css-brunch) \* [Grunt](http://gruntjs.com): [grunt-contrib-cssmin](https://github.com/gruntjs/grunt-contrib-cssmin) \* [Gulp](http://gulpjs.com/): [gulp-clean-css](https://github.com/scniro/gulp-clean-css) \* [Gulp](http://gulpjs.com/): [using vinyl-map as a wrapper - courtesy of @sogko](https://github.com/jakubpawlowicz/clean-css/issues/342) \* [component-builder2](https://github.com/component/builder2.js): [builder-clean-css](https://github.com/poying/builder-clean-css) \* [Metalsmith](http://metalsmith.io): [metalsmith-clean-css](https://github.com/aymericbeaumet/metalsmith-clean-css) \* [Lasso](https://github.com/lasso-js/lasso): [lasso-clean-css](https://github.com/yomed/lasso-clean-css) \* [Start](https://github.com/start-runner/start): [start-clean-css](https://github.com/start-runner/clean-css) ## How to use clean-css from web browser? \* https://jakubpawlowicz.github.io/clean-css/ (official web interface) \* http://refresh-sf.com/ \* http://adamburgess.github.io/clean-css-online/ # Contributing See [CONTRIBUTING.md](https://github.com/jakubpawlowicz/clean-css/blob/master/CONTRIBUTING.md). ## How to get started? First clone the sources: ```bash git clone git@github.com:jakubpawlowicz/clean-css.git ``` then install dependencies: ```bash cd clean-css npm install ``` then use any of the following commands to verify your copy: ```bash npm run bench # for clean-css benchmarks (see [test/bench.js](https://github.com/jakubpawlowicz/clean-css/blob/master/test/bench.js) for details) npm run browserify # to create the browser-ready clean-css version npm run check # to lint JS sources with [JSHint](https://github.com/jshint/jshint/) npm test # to run all tests ``` # Acknowledgments Sorted alphabetically by GitHub handle: \* [@abarre](https://github.com/abarre) (Anthony Barre) for improvements to `@import` processing; \* [@alexlamsl](https://github.com/alexlamsl) (Alex Lam S.L.) for testing early clean-css 4 versions, reporting bugs, and suggesting numerous improvements. \* [@altschuler](https://github.com/altschuler) (Simon Altschuler) for fixing `@import` processing inside comments; \* [@ben-eb](https://github.com/ben-eb) (Ben Briggs) for sharing ideas about CSS optimizations; \* [@davisjam](https://github.com/davisjam) (Jamie Davis) for disclosing ReDOS vulnerabilities; \* [@facelessuser](https://github.com/facelessuser) (Isaac) for pointing out a flaw in clean-css' stateless mode; \* [@grandrath](https://github.com/grandrath) (Martin Grandrath) for improving `minify` method source traversal in ES6; \* [@jmalonzo](https://github.com/jmalonzo) (Jan Michael Alonzo) for a patch removing node.js' old `sys` package; \* [@lukeapage](https://github.com/lukeapage) (Luke Page) for suggestions and testing the source maps feature; Plus everyone else involved in [#125](https://github.com/jakubpawlowicz/clean-css/issues/125) for pushing it forward; \* [@madwizard-thomas](https://github.com/madwizard-thomas) for sharing ideas about `@import` inlining and URL rebasing. \* [@ngyikp](https://github.com/ngyikp) (Ng Yik Phang) for testing early clean-css 4 versions, reporting bugs, and suggesting numerous improvements. \* [@wagenet](https://github.com/wagenet) (Peter Wagenet) for suggesting improvements to `@import` inlining behavior; \* [@venemo](https://github.com/venemo) (Timur Kristóf) for an outstanding contribution of advanced property optimizer for 2.2 release; \* [@vvo](https://github.com/vvo) (Vincent Voyer) for a patch with better empty element regex and for inspiring us to do many performance improvements in 0.4 release; \* [@xhmikosr](https://github.com/xhmikosr) for suggesting new features, like option to remove special comments and strip out URLs quotation, and pointing out numerous improvements like JSHint, media queries, etc. # License clean-css is released under the [MIT License](https://github.com/jakubpawlowicz/clean-css/blob/master/LICENSE).