# 6.1.2

* Fixed: erroneous trailing combinators in pseudos

# 6.1.1

* Fixed: improve typings of constructor helpers (#292)

# 6.1.0

* Feature: add sourceIndex to Selector nodes (#290)

# 6.0.16

* Fixed: add missing index argument to each/walk callback types (#289)

# 6.0.15

* Fixed: Node#prev and Node#next type for the first/last node

# 6.0.14

* Fixed: type definitions

# 6.0.13

* Fixed: throw on unexpected pipe symbols

# 6.0.12

* Fixed: clone arguments should be optional

# 6.0.11

* Fixed: parse attribute case insensitivity flag

# 6.0.10

* Fixed: isPseudoElement() supports :first-letter and :first-line

# 6.0.9

* Fixed: Combinator.raws property type

# 6.0.8

* Fixed: reduced size

# 6.0.7

* Fixed: parse animation percents

# 6.0.6

* Fixed: parse quoted attributes containing a newline correctly

# 6.0.5

* Perf: rework unesc for a 63+% performance boost

# 6.0.4

* Fixed: ts errors

# 6.0.3

* Fixed: replace node built-in "util" module with "util-deprecate"
* Fixed: handle uppercase pseudo elements
* Fixed: do not create invalid combinator before comment

# 6.0.2

* Fixed an issue with parsing and stringifying an empty attribute value

# 6.0.1

* Fixed an issue with unicode surrogate pair parsing

# 6.0.0

* Updated: cssesc to 3.0.0 (major)
* Fixed: Issues with escaped id and class selectors

# 5.0.0

* Allow escaped dot within class name.
* Update PostCSS to 7.0.7 (patch)

# 5.0.0-rc.4

* Fixed an issue where comments immediately after an insensitive (in attribute) were not parsed correctly.
* Updated cssesc to 2.0.0 (major).
* Removed outdated integration tests.
* Added tests for custom selectors, tags with attributes, the universal selector with pseudos, and tokens after combinators.

# 5.0.0-rc.1

To ease adoption of the v5.0 release, we have relaxed the node version check performed by npm at installation time to allow for node 4, which remains officially unsupported, but likely to continue working for the time being.

# 5.0.0-rc.0

This release has **BREAKING CHANGES** that were required to fix regressions in 4.0.0 and to make the Combinator Node API consistent for all combinator types. Please read carefully.

## Summary of Changes

* The way a descendent combinator that isn't a single space character (E.g. .a .b) is stored in the AST has changed.
* Named Combinators (E.g. .a /for/ .b) are now properly parsed as a combinator.
* It is now possible to look up a node based on the source location of a character in that node and to query nodes if they contain some character.
* Several bug fixes that caused the parser to hang and run out of memory when a / was encountered have been fixed.
* The minimum supported version of Node is now v6.0.0.

### Changes to the Descendent Combinator

In prior releases, the value of a descendant combinator with multiple spaces included all the spaces.

* .a .b: Extra spaces are now stored as space before.
  + Old & Busted:
    - combinator.value === " "
  + New hotness:
    - combinator.value === " " && combinator.spaces.before === " "
* .a /\*comment\*/.b: A comment at the end of the combinator causes extra space to become after space.
  + Old & Busted:
    - combinator.value === " "
    - combinator.raws.value === " /\*comment/"
  + New hotness:
    - combinator.value === " "
    - combinator.spaces.after === " "
    - combinator.raws.spaces.after === " /\*comment\*/"
* .a<newline>.b: whitespace that doesn't start or end with a single space character is stored as a raw value.
  + Old & Busted:
    - combinator.value === "\n"
    - combinator.raws.value === undefined
  + New hotness:
    - combinator.value === " "
    - combinator.raws.value === "\n"

### Support for "Named Combinators"

Although, nonstandard and unlikely to ever become a standard, combinators like /deep/ and /for/ are now properly supported.

Because they've been taken off the standardization track, there is no spec-official name for combinators of the form /<ident>/. However, I talked to [Tab Atkins](https://twitter.com/tabatkins) and we agreed to call them "named combinators" so now they are called that.

Before this release such named combinators were parsed without intention and generated three nodes of type "tag" where the first and last nodes had a value of "/".

* .a /for/ .b is parsed as a combinator.
  + Old & Busted:
    - root.nodes[0].nodes[1].type === "tag"
    - root.nodes[0].nodes[1].value === "/"
  + New hotness:
    - root.nodes[0].nodes[1].type === "combinator"
    - root.nodes[0].nodes[1].value === "/for/"
* .a /F\6fR/ .b escapes are handled and uppercase is normalized.
  + Old & Busted:
    - root.nodes[0].nodes[2].type === "tag"
    - root.nodes[0].nodes[2].value === "F\\6fR"
  + New hotness:
    - root.nodes[0].nodes[1].type === "combinator"
    - root.nodes[0].nodes[1].value === "/for/"
    - root.nodes[0].nodes[1].raws.value === "/F\\6fR/"

### Source position checks and lookups

A new API was added to look up a node based on the source location.

const selectorParser = require("postcss-selector-parser");

// You can find the most specific node for any given character

let combinator = selectorParser.astSync(".a > .b").atPosition(1,4);

combinator.toString() === " > ";

// You can check if a node includes a specific character

// Whitespace surrounding the node that is owned by that node

// is included in the check.

[2,3,4,5,6].map(column => combinator.isAtPosition(1, column));

// => [false, true, true, true, false]

# 4.0.0

This release has **BREAKING CHANGES** that were required to fix bugs regarding values with escape sequences. Please read carefully.

* **Identifiers with escapes** - CSS escape sequences are now hidden from the public API by default. The normal value of a node like a class name or ID, or an aspect of a node such as attribute selector's value, is unescaped. Escapes representing Non-ascii characters are unescaped into unicode characters. For example: bu\tton, .\31 00, #i\2764\FE0Fu, [attr="value is \"quoted\""] will parse respectively to the values button, 100, i❤️u, value is "quoted". The original escape sequences for these values can be found in the corresponding property name in node.raws. Where possible, deprecation warnings were added, but the nature of escape handling makes it impossible to detect what is escaped or not. Our expectation is that most users are neither expecting nor handling escape sequences in their use of this library, and so for them, this is a bug fix. Users who are taking care to handle escapes correctly can now update their code to remove the escape handling and let us do it for them.
* **Mutating values with escapes** - When you make an update to a node property that has escape handling The value is assumed to be unescaped, and any special characters are escaped automatically and the corresponding raws value is immediately updated. This can result in changes to the original escape format. Where the exact value of the escape sequence is important there are methods that allow both values to be set in conjunction. There are a number of new convenience methods for manipulating values that involve escapes, especially for attributes values where the quote mark is involved. See <https://github.com/postcss/postcss-selector-parser/pull/133> for an extensive write-up on these changes.

**Upgrade/API Example**

In 3.x there was no unescape handling and internal consistency of several properties was the caller's job to maintain. It was very easy for the developer to create a CSS file that did not parse correctly when some types of values were in use.

const selectorParser = require("postcss-selector-parser");

let attr = selectorParser.attribute({attribute: "id", operator: "=", value: "a-value"});

attr.value; // => "a-value"

attr.toString(); // => [id=a-value]

// Add quotes to an attribute's value.

// All these values have to be set by the caller to be consistent:

// no internal consistency is maintained.

attr.raws.unquoted = attr.value

attr.value = "'" + attr.value + "'";

attr.value; // => "'a-value'"

attr.quoted = true;

attr.toString(); // => "[id='a-value']"

In 4.0 there is a convenient API for setting and mutating values that may need escaping. Especially for attributes.

const selectorParser = require("postcss-selector-parser");

// The constructor requires you specify the exact escape sequence

let className = selectorParser.className({value: "illegal class name", raws: {value: "illegal\\ class\\ name"}});

className.toString(); // => '.illegal\\ class\\ name'

// So it's better to set the value as a property

className = selectorParser.className();

// Most properties that deal with identifiers work like this

className.value = "escape for me";

className.value; // => 'escape for me'

className.toString(); // => '.escape\\ for\\ me'

// emoji and all non-ascii are escaped to ensure it works in every css file.

className.value = "😱🦄😍";

className.value; // => '😱🦄😍'

className.toString(); // => '.\\1F631\\1F984\\1F60D'

// you can control the escape sequence if you want, or do bad bad things

className.setPropertyAndEscape('value', 'xxxx', 'yyyy');

className.value; // => "xxxx"

className.toString(); // => ".yyyy"

// Pass a value directly through to the css output without escaping it.

className.setPropertyWithoutEscape('value', '$REPLACE\_ME$');

className.value; // => "$REPLACE\_ME$"

className.toString(); // => ".$REPLACE\_ME$"

// The biggest changes are to the Attribute class

// passing quoteMark explicitly is required to avoid a deprecation warning.

let attr = selectorParser.attribute({attribute: "id", operator: "=", value: "a-value", quoteMark: null});

attr.toString(); // => "[id=a-value]"

// Get the value with quotes on it and any necessary escapes.

// This is the same as reading attr.value in 3.x.

attr.getQuotedValue(); // => "a-value";

attr.quoteMark; // => null

// Add quotes to an attribute's value.

attr.quoteMark = "'"; // This is all that's required.

attr.toString(); // => "[id='a-value']"

attr.quoted; // => true

// The value is still the same, only the quotes have changed.

attr.value; // => a-value

attr.getQuotedValue(); // => "'a-value'";

// deprecated assignment, no warning because there's no escapes

attr.value = "new-value";

// no quote mark is needed so it is removed

attr.getQuotedValue(); // => "new-value";

// deprecated assignment,

attr.value = "\"a 'single quoted' value\"";

// > (node:27859) DeprecationWarning: Assigning an attribute a value containing characters that might need to be escaped is deprecated. Call attribute.setValue() instead.

attr.getQuotedValue(); // => '"a \'single quoted\' value"';

// quote mark inferred from first and last characters.

attr.quoteMark; // => '"'

// setValue takes options to make manipulating the value simple.

attr.setValue('foo', {smart: true});

// foo doesn't require any escapes or quotes.

attr.toString(); // => '[id=foo]'

attr.quoteMark; // => null

// An explicit quote mark can be specified

attr.setValue('foo', {quoteMark: '"'});

attr.toString(); // => '[id="foo"]'

// preserves quote mark by default

attr.setValue('bar');

attr.toString(); // => '[id="bar"]'

attr.quoteMark = null;

attr.toString(); // => '[id=bar]'

// with no arguments, it preserves quote mark even when it's not a great idea

attr.setValue('a value \n that should be quoted');

attr.toString(); // => '[id=a\\ value\\ \\A\\ that\\ should\\ be\\ quoted]'

// smart preservation with a specified default

attr.setValue('a value \n that should be quoted', {smart: true, preferCurrentQuoteMark: true, quoteMark: "'"});

// => "[id='a value \\A that should be quoted']"

attr.quoteMark = '"';

// => '[id="a value \\A that should be quoted"]'

// this keeps double quotes because it wants to quote the value and the existing value has double quotes.

attr.setValue('this should be quoted', {smart: true, preferCurrentQuoteMark: true, quoteMark: "'"});

// => '[id="this should be quoted"]'

// picks single quotes because the value has double quotes

attr.setValue('a "double quoted" value', {smart: true, preferCurrentQuoteMark: true, quoteMark: "'"});

// => "[id='a "double quoted" value']"

// setPropertyAndEscape lets you do anything you want. Even things that are a bad idea and illegal.

attr.setPropertyAndEscape('value', 'xxxx', 'the password is 42');

attr.value; // => "xxxx"

attr.toString(); // => "[id=the password is 42]"

// Pass a value directly through to the css output without escaping it.

attr.setPropertyWithoutEscape('value', '$REPLACEMENT$');

attr.value; // => "$REPLACEMENT$"

attr.toString(); // => "[id=$REPLACEMENT$]"

# 3.1.2

* Fix: Removed dot-prop dependency since it's no longer written in es5.

# 3.1.1

* Fix: typescript definitions weren't in the published package.

# 3.1.0

* Fixed numerous bugs in attribute nodes relating to the handling of comments and whitespace. There's significant changes to attrNode.spaces and attrNode.raws since the 3.0.0 release.
* Added Attribute#offsetOf(part) to get the offset location of attribute parts like "operator" and "value". This is most often added to Attribute#sourceIndex for error reporting.

# 3.0.0

## Breaking changes

* Some tweaks to the tokenizer/attribute selector parsing mean that whitespace locations might be slightly different to the 2.x code.
* Better attribute selector parsing with more validation; postcss-selector-parser no longer uses regular expressions to parse attribute selectors.
* Added an async API (thanks to @jacobp100); the default process API is now async, and the sync API is now accessed through processSync instead.
* process() and processSync() now return a string instead of the Processor instance.
* Tweaks handling of Less interpolation (thanks to @jwilsson).
* Removes support for Node 0.12.

## Other changes

* ast() and astSync() methods have been added to the Processor. These return the Root node of the selectors after processing them.
* transform() and transformSync() methods have been added to the Processor. These return the value returned by the processor callback after processing the selectors.
* Set the parent when inserting a node (thanks to @chriseppstein).
* Correctly adjust indices when using insertBefore/insertAfter (thanks to @tivac).
* Fixes handling of namespaces with qualified tag selectors.
* process, ast and transform (and their sync variants) now accept a postcss rule node. When provided, better errors are generated and selector processing is automatically set back to the rule selector (unless the updateSelector option is set to false.)
* Now more memory efficient when tokenizing selectors.

### Upgrade hints

The pattern of:

rule.selector = processor.process(rule.selector).result.toString();

is now:

processor.processSync(rule)

# 2.2.3

* Resolves an issue where the parser would not reduce multiple spaces between an ampersand and another simple selector in lossy mode (thanks to @adam-26).

# 2.2.2

* No longer hangs on an unescaped semicolon; instead the parser will throw an exception for these cases.

# 2.2.1

* Allows a consumer to specify whitespace tokens when creating a new Node (thanks to @Semigradsky).

# 2.2.0

* Added a new option to normalize whitespace when parsing the selector string (thanks to @adam-26).

# 2.1.1

* Better unquoted value handling within attribute selectors (thanks to @evilebottnawi).

# 2.1.0

* Added: Use string constants for all node types & expose them on the main parser instance (thanks to @Aweary).

# 2.0.0

This release contains the following breaking changes:

* Renamed all eachInside iterators to walk. For example, eachTag is now walkTags, and eachInside is now walk.
* Renamed Node#removeSelf() to Node#remove().
* Renamed Container#remove() to Container#removeChild().
* Renamed Node#raw to Node#raws (thanks to @davidtheclark).
* Now parses & as the *nesting* selector, rather than a *tag* selector.
* Fixes misinterpretation of Sass interpolation (e.g. #{foo}) as an id selector (thanks to @davidtheclark).

and;

* Fixes parsing of attribute selectors with equals signs in them (e.g. [data-attr="foo=bar"]) (thanks to @montmanu).
* Adds quoted and raw.unquoted properties to attribute nodes (thanks to @davidtheclark).

# 1.3.3

* Fixes an infinite loop on ) and ] tokens when they had no opening pairs. Now postcss-selector-parser will throw when it encounters these lone tokens.

# 1.3.2

* Now uses plain integers rather than str.charCodeAt(0) for compiled builds.

# 1.3.1

* Update flatten to v1.x (thanks to @shinnn).

# 1.3.0

* Adds a new node type, String, to fix a crash on selectors such as foo:bar("test").

# 1.2.1

* Fixes a crash when the parser encountered a trailing combinator.

# 1.2.0

* A more descriptive error is thrown when the parser expects to find a pseudo-class/pseudo-element (thanks to @ashelley).
* Adds support for line/column locations for selector nodes, as well as a Node#sourceIndex method (thanks to @davidtheclark).

# 1.1.4

* Fixes a crash when a selector started with a > combinator. The module will now no longer throw if a selector has a leading/trailing combinator node.

# 1.1.3

* Fixes a crash on @ tokens.

# 1.1.2

* Fixes an infinite loop caused by using parentheses in a non-pseudo element context.

# 1.1.1

* Fixes a crash when a backslash ended a selector string.

# 1.1.0

* Adds support for replacing multiple nodes at once with replaceWith (thanks to @jonathantneal).
* Parser no longer throws on sequential IDs and trailing commas, to support parsing of selector hacks.

# 1.0.1

* Fixes using insertAfter and insertBefore during iteration.

# 1.0.0

* Adds clone and replaceWith methods to nodes.
* Adds insertBefore and insertAfter to containers.
* Stabilises API.

# 0.0.5

* Fixes crash on extra whitespace inside a pseudo selector's parentheses.
* Adds sort function to the container class.
* Enables the parser to pass its input through without transforming.
* Iteration-safe each and eachInside.

# 0.0.4

* Tidy up redundant duplication.
* Fixes a bug where the parser would loop infinitely on universal selectors inside pseudo selectors.
* Adds length getter and eachInside, map, reduce to the container class.
* When a selector has been removed from the tree, the root node will no longer cast it to a string.
* Adds node type iterators to the container class (e.g. eachComment).
* Adds filter function to the container class.
* Adds split function to the container class.
* Create new node types by doing parser.id(opts) etc.
* Adds support for pseudo classes anywhere in the selector.

# 0.0.3

* Adds next and prev to the node class.
* Adds first and last getters to the container class.
* Adds every and some iterators to the container class.
* Add empty alias for removeAll.
* Combinators are now types of node.
* Fixes the at method so that it is not an alias for index.
* Tidy up creation of new nodes in the parser.
* Refactors how namespaces are handled for consistency & less redundant code.
* Refactors AST to use nodes exclusively, and eliminates excessive nesting.
* Fixes nested pseudo parsing.
* Fixes whitespace parsing.

# 0.0.2

* Adds support for namespace selectors.
* Adds support for selectors joined by escaped spaces - such as .\31\ 0.

# 0.0.1

* Initial release.