CSS Nesting Module

W3C First Public Working Draft, 31 August 2021

This version:
https://www.w3.org/TR/2021/WD-css-nesting-1-20210831/

Latest published version:
https://www.w3.org/TR/css-nesting-1/

Editor's Draft:
https://drafts.csswg.org/css-nesting/

Issue Tracking:
CSSWG Issues Repository

Editors:
Tab Atkins-Bittner (Google)
Adam Argyle (Google)

Suggest an Edit for this Spec:
GitHub Editor

Abstract

This module introduces the ability to nest one style rule inside another, with the selector of the child rule relative to the selector of the parent rule. This increases the modularity and maintainability of CSS stylesheets.

CSS is a language for describing the rendering of structured documents (such as HTML and XML) on screen, on paper, etc.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document was published by the CSS Working Group as a **First Public Working Draft**. Publication as a First Public Working Draft does not imply endorsement by the W3C Membership.

This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Please send feedback by filing issues in GitHub (preferred), including the spec code “css-nesting” in the title, like this: “[css-nesting] …*summary of comment*…”. All issues and comments are archived. Alternately, feedback can be sent to the (archived) public mailing list www-style@w3.org.

This document is governed by the 15 September 2020 W3C Process Document.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

§ 1. Introduction

This section is not normative.

This module describes support for nesting a style rule within another style rule, allowing the inner rule's selector to reference the elements matched by the outer rule. This feature allows related styles to be aggregated into a single structure within the CSS document, improving readability and maintainability.

§ 1.1. Module Interactions

This module introduces new parser rules that extend the [CSS21] parser model. This module introduces selectors that extend the [SELECTORS4] module.

§ 1.2. Values

This specification does not define any new properties or values.

§ 1.3. Motivation

The CSS for even moderately complicated web pages often include lots of duplication for the purpose of styling related content. For example, here is a portion of the CSS markup for one version of the [CSS-COLOR-3] module:

EXAMPLE 1

```
table.colortable td {
  text-align:center;
}
table.colortable td.c {
  text-transform:uppercase;
}
table.colortable td:first-child, table.colortable td:first-child+td {
  border:1px solid black;
}
table.colortable th {
  text-align:center;
  background:black;
  color:white;
}
```

Nesting allows the grouping of related style rules, like this:

EXAMPLE 2

```
table.colortable {
  & td {
    text-align:center;
    &.c { text-transform:uppercase }
    &:first-child, &:first-child + td { border:1px solid black }
  }
  & th {
    text-align:center;
    background:black;
    color:white;
  }
}
```

Besides removing duplication, the grouping of related rules improves the readability and maintainability of the resulting CSS.

§ 2. Nesting Style Rules

Style rules can be nested inside of other styles rules. These ***nested style rules*** act exactly like ordinary style rules—associating properties with elements via selectors—but they "inherit" their parent rule's selector context, allowing them to further build on the parent's selector without having to repeat it, possibly multiple times.

There are two closely-related syntaxes for creating nested style rules:

- Direct nesting, where the nested style rule is written normally inside the parent rule, but with the requirement that the nested style rule's selector is nest-prefixed.
- The '@nest' rule, which imposes less constraints on the nested style rule's selector.

Aside from the slight difference in how they're written, both methods are exactly equivalent in functionality.

▶ **Why can't everything be directly nested?**

§ 2.1. Direct Nesting

A style rule can be ***directly nested*** within another style rule if its selector is nest-prefixed.

To be ***nest-prefixed***, a nesting selector must be the first simple selector in the first compound selector of the selector. If the selector is a list of selectors, every complex selector in the list must be nest-prefixed for the selector as a whole to be nest-prefixed.

EXAMPLE 3

For example, the following nestings are valid:

```
/* & can be used on its own */
.foo {
  color: blue;
  & > .bar { color: red; }
}
/* equivalent to
  .foo { color: blue; }
  .foo > .bar { color: red; }
*/


/* or in a compound selector,
   refining the parent's selector */
.foo {
  color: blue;
  &.bar { color: red; }
}
/* equivalent to
  .foo { color: blue; }
  .foo.bar { color: red; }
*/

/* multiple selectors in the list must all
   start with & */
.foo, .bar {
  color: blue;
  & + .baz, &.qux { color: red; }
}
/* equivalent to
  .foo, .bar { color: blue; }
  :is(.foo, .bar) + .baz,
  :is(.foo, .bar).qux { color: red; }
*/

/* & can be used multiple times in a single selector */
.foo {
  color: blue;
  & .bar & .baz & .qux { color: red; }
}
/* equivalent to
```

```
  .foo { color: blue; }
  .foo .bar .foo .baz .foo .qux { color: red; }
*/

/* Somewhat silly, but can be used all on its own, as well. */
.foo {
  color: blue;
  & { padding: 2ch; }
}
/* equivalent to
  .foo { color: blue; }
  .foo { padding: 2ch; }

  // or

  .foo {
    color: blue;
    padding: 2ch;
  }
*/

/* Again, silly, but can even be doubled up. */
.foo {
  color: blue;
  && { padding: 2ch; }
}
/* equivalent to
  .foo { color: blue; }
  .foo.foo { padding: 2ch; }
*/

/* The parent selector can be arbitrarily complicated */
.error, #404 {
  &:hover > .baz { color: red; }
}
/* equivalent to
  :is(.error, #404):hover > .baz { color: red; }
*/

/* As can the nested selector */
.foo {
  &:is(.bar, &.baz) { color: red; }
}
```

```
/* equivalent to
  .foo:is(.bar, .foo.baz) { color: red; }
*/

/* Multiple levels of nesting "stack up" the selectors */
figure {
  margin: 0;

  & > figcaption {
    background: hsl(0 0% 0% / 50%);

    & > p {
      font-size: .9rem;
    }
  }
}
/* equivalent to
  figure { margin: 0; }
  figure > figcaption { background: hsl(0 0% 0% / 50%); }
  figure > figcaption > p { font-size: .9rem; }
*/
```

But these are not:

```
/* No & at all */
.foo {
  color: blue;
  .bar {
    color: red;
  }
}

/* & isn't the first simple selector */
.foo {
  color: blue;
  .bar& {
    color: red;
  }
}

/* & isn't the first selector of every one in the list */
.foo, .bar {
```

```
  color: blue;
  & + .baz, .qux { color: red; }
}
```

The last example here isn't technically ambiguous, since the selector as a whole does start with an '&', but it's an editing hazard—if the rule is refactored to remove the first selector or rearrange the selectors in the list, which normally would always remain valid, it would result in a now-ambiguous invalid selector.

Some CSS-generating tools will concatenate selectors like strings, allowing authors to build up a single simple selector across nesting levels. This is sometimes used by selector-organization methods like BEM to reduce repetition across a file, when the selectors themselves have significant repetition internally.

For example, if one component uses the class '.foo', and a nested component uses '.foo__bar', you could write this in Sass as:

```
.foo {
  color: blue;
  &__bar { color: red; }
}
/* In Sass, this is equivalent to
   .foo { color: blue; }
   .foo__bar { color: red; }
*/
```

Unfortunately, this method is inconsistent with selector syntax in general, and at best requires heuristics tuned to particularly selector-writing practices to recognize when the author wants it, versus the author attempting to add a type selector in the nested rule. '__bar', for example, *is* a valid custom element name in HTML.

As such, CSS can't do this; the nested selector components are interpreted on their own, and not "concatenated":

```
.foo {
  color: blue;
  &__bar { color: red; }
}
/* In CSS, this is instead equivalent to
   .foo { color: blue; }
   __bar.foo { color: red; }
*/
```

§ 2.2. The Nesting At-Rule: '@nest'

While direct nesting looks nice, it is somewhat fragile. Some valid nesting selectors, like '.foo &', are disallowed, and editing the selector in certain ways can make the rule invalid unexpectedly. As well,

some authors find the nesting challenging to distinguish visually from the surrounding declarations.

To aid in all these issues, this specification defines the ‘@nest’ rule, which imposes fewer restrictions on how to validly nest style rules. Its syntax is:

@nest = @nest <selector-list> { <style-block> }

The ‘@nest’ rule is only valid inside of a style rule. If used in any other context (particularly, at the top-level of a stylesheet) the rule is invalid.

The ‘@nest’ rule functions identically to a nested style rule: it starts with a selector, and contains a block of declarations that apply to the elements the selector matches. That block is treated identically to a style rule’s block, so anything valid in a style rule (such as additional ‘@nest’ rules) is also valid here.

The only difference between ‘@nest’ and a directly nested style rule is that the selector used in a ‘@nest’ rule is less constrained: it only must be ***nest-containing***, which means it contains a nesting selector in it *somewhere*, rather than requiring it to be at the start of each selector. A list of selectors is nest-containing if all of its individual complex selectors are nest-containing.

EXAMPLE 4

Anything you can do with direct nesting, you can do with an '@nest' rule, so the following is valid:

```
.foo {
  color: red;
  @nest & > .bar {
    color: blue;
  }
}
/* equivalent to
  .foo { color: red; }
  .foo > .bar { color: blue; }
*/
```

But '@nest' allows selectors that don't start with an '&', so the following are also valid:

```
.foo {
  color: red;
  @nest .parent & {
    color: blue;
  }
}
/* equivalent to
  .foo { color: red; }
  .parent .foo { color: blue; }
*/

.foo {
  color: red;
  @nest :not(&) {
    color: blue;
  }
}
/* equivalent to
  .foo { color: red; }
  :not(.foo) { color: blue; }
*/
```

But the following are invalid:

```
.foo {
  color: red;
  @nest .bar {
    color: blue;
  }
}
/* Invalid because there's no nesting selector */

.foo {
  color: red;
  @nest & .bar, .baz {
    color: blue;
  }
}
/* Invalid because not all selectors in the list
  contain a nesting selector */
```

EXAMPLE 5

Directly nested style rules and '@nest' rules can be arbitrarily mixed. For example:

```
.foo {
  color: blue;
  @nest .bar & {
    color: red;
    &.baz {
      color: green;
    }
  }
}
/* equivalent to
  .foo { color: blue; }
  .bar .foo { color: red; }
  .bar .foo.baz { color: green; }
```

§ 2.3. Nesting Conditional Rules

In addition to '@nest' rules and directly nested style rules, this specification allows ***nested conditional group rules*** inside of style rules.

When nested in this way, the contents of a conditional group rule are parsed as <style-block> rather than <stylesheet>:

- Properties can be directly used, applying to the same elements as the parent rule (when the conditional group rule matches)
- Style rules are treated as directly nested, and so must have nest-prefixed selectors, with their nesting selector taking its definition from the nearest ancestor style rule.
- '@nest' rules are allowed, again with their nesting selector taking its definition from the nearest ancestor style rule.

> Note: This implies that "normal" style rules, without a nesting selector, are invalid in a nested conditional group rule.

EXAMPLE 6

For example, the following conditional nestings are valid:

```
/* Properties can be directly used */
.foo {
  display: grid;

  @media (orientation: landscape) {
    grid-auto-flow: column;
  }
}
/* equivalent to
  .foo { display: grid; }

  @media (orientation: landscape) {
    & {
      grid-auto-flow: column;
    }
  }
*/

/* finally equivalent to
  .foo { display: grid; }

  @media (orientation: landscape) {
    .foo {
      grid-auto-flow: column;
    }
  }
*/

/* Conditionals can be further nested */
.foo {
  display: grid;

  @media (orientation: landscape) {
    grid-auto-flow: column;

    @media (min-inline-size > 1024px) {
      max-inline-size: 1024px;
    }
  }
}
```

```
/* equivalent to
  .foo { display: grid; }

  @media (orientation: landscape) {
    .foo {
      grid-auto-flow: column;
    }
  }

  @media (orientation: landscape) and (min-inline-size > 1024px) {
    .foo {
      max-inline-size: 1024px;
    }
  }
*/
```

But the following are invalid:

```
.foo {
  color: red;

  @media (min-width: 480px) {
    & h1, h2 { color: blue; }
  }
}
/* Invalid because not all selectors in the list
  contain a nesting selector */

.foo {
  color: red;

  @nest @media (min-width: 480px) {
    & { color: blue; }
  }
}
/* Invalid because @nest expects a selector prelude,
  instead a conditional group rule was provided */
```

§ 2.4. Mixing Nesting Rules and Declarations

When a style rule contains both declarations and nested style rules or nested conditional group rules, the declarations must come first, followed by the nested rules. Declarations occuring *after* a nested rule are invalid and ignored.

EXAMPLE 7

For example, in the following code:

```
article {
  color: green;
  & { color: blue; }
  color: red;
}
```

The 'color: red' declaration is invalid and ignored, since it occurs after the nested style rule.

However, further nested rules are still valid, as in this example:

```
article {
  color: green;
  & { color: blue; }
  color: red;
  &.foo { color: yellow; } /* valid! */
}
```

For the purpose of determining the Order Of Appearance, nested style rules and nested conditional group rules are considered to come *after* their parent rule.

For example:

```
article {
  color: blue;
  & { color: red; }
}
```

Both declarations have the same specificity (0,0,1), but the nested rule is considered to come *after* its parent rule, so the 'color: red' declarations wins the cascade.

On the other hand, in this example:

```
article {
  color: blue;
  @nest :where(&) { color: red; }
}
```

The ‘:where()’ pseudoclass reduces the specificity of the nesting selector to 0, so the ‘color: red’ declaration now has a specificity of (0,0,0), and loses to the ‘color: blue’ declaration before "Order Of Appearance" comes into consideration.

§ 3. Nesting Selector: the ‘&’ selector

When using a nested style rule, one must be able to refer to the elements matched by the parent rule; that is, after all, *the entire point of nesting*. To accomplish that, this specification defines a new selector, the ***nesting selector***, written as ***‘&’*** (U+0026 AMPERSAND).

When used in the selector of a nested style rule, the nesting selector represents the elements matched by the parent rule. When used in any other context, it represents nothing. (That is, it’s valid, but matches no elements.)

> The nesting selector can be desugared by replacing it with the parent style rule’s selector, wrapped in an ‘:is()’ selector. For example,
>
> ```
> a, b {
>   & c { color: blue; }
> }
> ```
>
> is equivalent to
>
> ```
> :is(a, b) c { color: blue; }
> ```

The specificity of the nesting selector is equal to the largest specificity among the complex selectors in the parent style rule’s selector list (identical to the behavior of ‘:is()’).

EXAMPLE 8

For example, given the following style rules:

```
#a, b {
  & c { color: blue; }
}
.foo c { color: red; }
```

Then in a DOM structure like

```
<b class=foo>
  <c>Blue text</c>
</b>
```

The text will be blue, rather than red. The specificity of the '&' is the larger of the specificities of '#a' ([1,0,0]) and 'b' ([0,0,1]), so it's [1,0,0], and the entire '& c' selector thus has specificity [1,0,1], which is larger than the specificity of '.foo c' ([0,1,1]).

Notably, this is *different* than the result you'd get if the nesting were manually expanded out into non-nested rules, since the 'color: blue' declaration would then be matching due to the 'b c' selector ([0,0,2]) rather than '#a c' ([1,0,1]).

▶ **Why is the specificity different than non-nested rules?**

The nesting selector is allowed anywhere in a compound selector, even before a type selector, violating the normal restrictions on ordering within a compound selector.

EXAMPLE 9

For example, '&div' is a valid nesting selector, meaning "whatever the parent rules matches, but only if it's also a `<div>` element".

It could also be written as 'div&' with the same meaning, but that wouldn't be valid to start a directly nested style rule.

§ 4. CSSOM

§ 4.1. Modifications to CSSStyleRule

CSS style rules gain the ability to have nested rules:

```
partial interface CSSStyleRule {
  [SameObject] readonly attribute CSSRuleList cssRules;
  unsigned long insertRule(CSSOMString rule, optional unsigned long index = 0);
  undefined deleteRule(unsigned long index);
};
```

The ***cssRules*** attribute must return a CSSRuleList object for the child CSS rules.

The ***insertRule(rule, index)*** method must return the result of invoking insert a CSS rule *rule* into the child CSS rules at *index*.

The ***deleteRule(index)*** method must remove a CSS rule from the child CSS rules at *index*.

§ 4.2. The CSSNestingRule Interface

The CSSNestingRule interfaces represents a '@nest' rule:

```
[Exposed=Window]
interface CSSNestingRule : CSSRule {
  attribute CSSOMString selectorText;
  [SameObject, PutForwards=cssText] readonly attribute CSSStyleDeclaration style;
  [SameObject] readonly attribute CSSRuleList cssRules;
  unsigned long insertRule(CSSOMString rule, optional unsigned long index = 0);
  undefined deleteRule(unsigned long index);
};
```

The ***selectorText*** attribute, on getting, must return the result of serializing the associated selector list. On setting the selectorText attribute these steps must be run:

1. Run the parse a group of selectors algorithm on the given value.
2. If the algorithm returns a non-null value replace the associated selector list with the returned value.
3. Otherwise, if the algorithm returns a null value, do nothing.

The ***style*** attribute must return a `CSSStyleDeclaration` object for the style rule, with the following properties:

computed flag
: Unset.

declarations
: The declared declarations in the rule, in specified order.

parent CSS rule
: The this object.

owner node
: Null.

The ***cssRules*** attribute must return a `CSSRuleList` object for the child CSS rules.

The ***insertRule(rule, index)*** method must return the result of invoking insert a CSS rule *rule* into the child CSS rules at *index*.

The ***deleteRule(index)*** method must remove a CSS rule from the child CSS rules at *index*.

To serialize a `CSSNestingRule`: return the result of the following steps:

1. Let *s* initially be the string "`@nest`" followed by a single SPACE (U+0020).
2. Append to *s* the result of performing serialize a group of selectors on the rule's associated selectors, followed by the string " {", i.e., a single SPACE (U+0020), followed by LEFT CURLY BRACKET (U+007B).
3. Let *decls* be the result of performing serialize a CSS declaration block on the rule's associated declarations, or null if there are no such declarations.
4. Let *rules* be the result of performing serialize a CSS rule on each rule in the rule's `cssRules` list, or null if there are no such rules.
5. If *decls* and *rules* are both null, append " }" to *s* (i.e. a single SPACE (U+0020) followed by RIGHT CURLY BRACKET (U+007D)) and return *s*.
6. If *rules* is null:
    1. Append a single SPACE (U+0020) to *s*
    2. Append *decls* to *s*
    3. Append " }" to *s* (i.e. a single SPACE (U+0020) followed by RIGHT CURLY BRACKET (U+007D)).
    4. Return *s*.

7. Otherwise:
    1. If *decls* is not null, prepend it to *rules*.
    2. For each *rule* in *rules*:
        1. Append a newline followed by two spaces to *s*.
        2. Append *rule* to *s*.
    3. Append a newline followed by RIGHT CURLY BRACKET (U+007D) to *s*.
    4. Return *s*.

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

> EXAMPLE 10
>
> This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

> Note, this is an informative note.

Advisements are normative sections styled to evoke special attention and are set apart from other normative text with `<strong class="advisement">`, like this:

UAs MUST provide an accessible alternative.

§ Conformance classes

Conformance to this specification is defined for three conformance classes:

style sheet
A CSS style sheet.

renderer
A UA that interprets the semantics of a style sheet and renders documents that use them.

authoring tool
A UA that writes a style sheet.

A style sheet is conformant to this specification if all of its statements that use syntax defined in this module are valid according to the generic CSS grammar and the individual grammars of each feature defined in this module.

A renderer is conformant to this specification if, in addition to interpreting the style sheet as defined by the appropriate specifications, it supports all the features defined by this specification by parsing them correctly and rendering the document accordingly. However, the inability of a UA to correctly render a document due to limitations of the device does not make the UA non-conformant. (For example, a UA is not required to render color on a monochrome monitor.)

An authoring tool is conformant to this specification if it writes style sheets that are syntactically correct according to the generic CSS grammar and the individual grammars of each feature in this module, and meet all other conformance requirements of style sheets as described in this module.

§ Partial implementations

So that authors can exploit the forward-compatible parsing rules to assign fallback values, CSS renderers **must** treat as invalid (and ignore as appropriate) any at-rules, properties, property values, keywords, and other syntactic constructs for which they have no usable level of support. In particular, user agents **must not** selectively ignore unsupported component values and honor supported values in a single multi-value property declaration: if any value is considered invalid (as unsupported values must be), CSS requires that the entire declaration be ignored.

§ Implementations of Unstable and Proprietary Features

To avoid clashes with future stable CSS features, the CSSWG recommends following best practices for the implementation of unstable features and proprietary extensions to CSS.

§ Non-experimental implementations

Once a specification reaches the Candidate Recommendation stage, non-experimental implementations are possible, and implementors should release an unprefixed implementation of any CR-level feature they can demonstrate to be correctly implemented according to spec.

To establish and maintain the interoperability of CSS across implementations, the CSS Working Group requests that non-experimental CSS renderers submit an implementation report (and, if necessary, the testcases used for that implementation report) to the W3C before releasing an unprefixed implementation of any CSS features. Testcases submitted to W3C are subject to review and correction by the CSS Working Group.

Further information on submitting testcases and implementation reports can be found from on the CSS Working Group's website at https://www.w3.org/Style/CSS/Test/. Questions should be directed to the public-css-testsuite@w3.org mailing list.

§ Index

§ Terms defined by this specification

nest-prefixed, in § 2.1

selectorText, in § 4.2

style, in § 4.2

§ Terms defined by reference

[css-color-4] defines the following terms:

- color

[css-conditional-3] defines the following terms:

- conditional group rule

[css-syntax-3] defines the following terms:

- <stylesheet>
- style rule

[cssom-1] defines the following terms:

- CSSOMString
- CSSRule
- CSSRuleList
- CSSStyleDeclaration
- CSSStyleRule
- child css rules
- computed flag
- cssText
- declarations
- insert a css rule
- owner node
- parent css rule
- parse a group of selectors
- remove a css rule
- selectorText
- serialize a css declaration block
- serialize a css rule
- serialize a group of selectors
- specified order

[HTML] defines the following terms:

- div

[SELECTORS4] defines the following terms:

- :is()
- :where()
- <selector-list>
- complex selector
- compound selector
- selector list
- simple selector
- specificity
- type selector

[WebIDL] defines the following terms:

- Exposed
- PutForwards
- SameObject
- this
- undefined
- unsigned long

§ References

§ Normative References

[CSS-COLOR-4]
Tab Atkins Jr.; Chris Lilley. *CSS Color Module Level 4*. 1 June 2021. WD. URL: https://www.w3.org/TR/css-color-4/

[CSS-CONDITIONAL-3]
David Baron; Elika Etemad; Chris Lilley. *CSS Conditional Rules Module Level 3*. 8 December 2020. CR. URL: https://www.w3.org/TR/css-conditional-3/

[CSS-SYNTAX-3]
Tab Atkins Jr.; Simon Sapin. *CSS Syntax Module Level 3*. 16 July 2019. CR. URL: https://www.w3.org/TR/css-syntax-3/

[CSS21]
Bert Bos; et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. 7 June 2011. REC. URL: https://www.w3.org/TR/CSS21/

[CSSOM-1]
Daniel Glazman; Emilio Cobos Álvarez. *CSS Object Model (CSSOM)*. 26 August 2021. WD. URL: https://www.w3.org/TR/cssom-1/

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://datatracker.ietf.org/doc/html/rfc2119

[SELECTORS4]
Elika Etemad; Tab Atkins Jr.. *Selectors Level 4*. 21 November 2018. WD. URL: https://www.w3.org/TR/selectors-4/

[WebIDL]
Boris Zbarsky. *Web IDL*. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

§ Informative References

[CSS-COLOR-3]
Tantek Çelik; Chris Lilley; David Baron. *CSS Color Module Level 3*. 5 August 2021. REC. URL: https://www.w3.org/TR/css-color-3/

[HTML]
Anne van Kesteren; et al. *HTML Standard*. Living Standard. URL: https://html.spec.whatwg.org/multipage/

§ IDL Index

```
partial interface CSSStyleRule {
  [SameObject] readonly attribute CSSRuleList cssRules;
  unsigned long insertRule(CSSOMString rule, optional unsigned long index = 0);
  undefined deleteRule(unsigned long index);
};

[Exposed=Window]
interface CSSNestingRule : CSSRule {
  attribute CSSOMString selectorText;
  [SameObject, PutForwards=cssText] readonly attribute CSSStyleDeclaration style;
  [SameObject] readonly attribute CSSRuleList cssRules;
  unsigned long insertRule(CSSOMString rule, optional unsigned long index = 0);
  undefined deleteRule(unsigned long index);
};
```