

CSS Cascading and Inheritance Level 4

W3C Candidate Recommendation Snapshot, 13 January 2022



▼ More details about this document

This version:

<https://www.w3.org/TR/2022/CR-css-cascade-4-20220113/>

Latest published version:

<https://www.w3.org/TR/css-cascade-4/>

Editor's Draft:

<https://drafts.csswg.org/css-cascade-4/>

Previous Versions:

<https://www.w3.org/TR/2021/WD-css-cascade-4-20211015/>

<https://www.w3.org/TR/2021/WD-css-cascade-4-20210319/>

<https://www.w3.org/TR/2020/WD-css-cascade-4-20200818/>

<https://www.w3.org/TR/2018/CR-css-cascade-4-20180828/>

<https://www.w3.org/TR/2016/CR-css-cascade-4-20160114/>

<https://www.w3.org/TR/2015/WD-css-cascade-4-20150908/>

<https://www.w3.org/TR/2015/WD-css-cascade-4-20150421/>

<https://www.w3.org/TR/2013/WD-css-cascade-3-20130730/>

<https://www.w3.org/TR/2013/WD-css3-cascade-20130103/>

<https://www.w3.org/TR/2005/WD-css3-cascade-20051215/>

History:

<https://www.w3.org/standards/history/css-cascade-4>

Implementation Report:

<https://wpt.fyi/results/css/css-cascade>

Feedback:

[CSSWG Issues Repository](#)

Editors:

[Elika J. Etemad / fantasai](#) (Invited Expert)

[Tab Atkins Jr.](#) (Google)

Suggest an Edit for this Spec:

[GitHub Editor](#)

Copyright © 2022 W3C® (MIT, ERCIM, Keio, Beihang). W3C [liability](#), [trademark](#) and [permissive document license](#) rules apply.

Abstract

This CSS module describes how to collate style rules and assign values to all properties on all elements. By way of cascading and inheritance, values are propagated for all properties on all elements.

New in this level are the ‘[revert](#)’ keyword and [<supports-condition>](#) for the ‘[@import](#)’ rule.

[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. 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/](https://www.w3.org/TR/css-cascade-4/).

This document was published by the [CSS Working Group](#) as a **Candidate Recommendation Snapshot** [Recommendation track](#). Publication as a Candidate Recommendation does not imply endorsement by W3C Members. A Candidate Recommendation Snapshot has received [wide review](#), is intended to gather implementation experience, and has commitments from Working Group members to [royalty-free licensing](#) for implementation. This document is intended to become a W3C Recommendation; it will remain a Candidate Recommendation until March 2022 to gather additional feedback.

Please send feedback by [filing issues in GitHub](#) (preferred), including the spec code “css-cascade” in the “[css-cascade] ...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 [2 November 2021 W3C Process Document](#).

This document was produced by a group operating under the [W3C Patent Policy](#). W3C maintains a [public disclosure](#) 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](#).

Support:		
☐	Android Browser	96+
☐	BlackBerry Browser	None
☐	BlackBerry Browser	None
☐	Chrome	84+
☐	Chrome for Android	96+
☐	Edge	84+
☐	Firefox	67+
☐	Firefox for Android	95+
☐	IE	None
☐	IE Mobile	None
☐	KaiOS Browser	None
☐	Opera	73+
☐	Opera Mini	None
☐	Opera Mobile	64+
☐	QQ Browser	None
☐	Safari	9.1+
☐	Safari on iOS	9.3+
☐	Samsung Internet	14.0+
☐	UC Browser for Android	None
Source: caniuse.com as of 2022-01-08		

Table of Contents

1	Introduction
1.1	Module Interactions
2	Importing Style Sheets: the ‘@import’ rule
2.1	Conditional ‘@import’ Rules
2.2	Processing Stylesheet Imports
2.3	Content-Type of CSS Style Sheets
3	Shorthand Properties
3.1	Property Aliasing
3.2	Resetting All Properties: the ‘all’ property
4	Value Processing
4.1	Declared Values
4.1.1	Value Aliasing
4.2	Cascaded Values
4.3	Specified Values
4.4	Computed Values
4.5	Used Values
4.5.1	Applicable Properties
4.6	Actual Values
4.7	Examples
5	Filtering
6	Cascading
6.1	Cascade Sorting Order
6.2	Cascading Origins
6.3	Important Declarations: the ‘!important’ annotation
6.4	Precedence of Non-CSS Presentational Hints

7	Defaulting
7.1	Initial Values
7.2	Inheritance
7.3	Explicit Defaulting
7.3.1	Resetting a Property: the ‘ initial ’ keyword
7.3.2	Explicit Inheritance: the ‘ inherit ’ keyword
7.3.3	Erasing All Declarations: the ‘ unset ’ keyword
7.3.4	Rolling Back Cascade Origins: the ‘ revert ’ keyword
8	Changes
8.1	Changes since the 15 Oct 2021 Working Draft
8.2	Changes Since the 28 August 2018 Candidate Recommendation
8.3	Changes Since the 14 January 2016 Candidate Recommendation
8.4	Changes Since the 21 April 2015 Working Draft
8.5	Additions Since Level 3
8.6	Additions Since Level 2

Acknowledgments

Privacy and Security Considerations

Conformance

Document conventions

Conformance classes

Partial implementations

Implementations of Unstable and Proprietary Features

Non-experimental implementations

CR exit criteria

Index

Terms defined by this specification

Terms defined by reference

References

Normative References

Informative References

Property Index

§ 1. Introduction

CSS defines a finite set of parameters, called *properties*, that direct the rendering of a document. Each [property](#) has a name (e.g., ‘[color](#)’, ‘[font-size](#)’, or ‘[border-style](#)’), a value space (e.g., [<color>](#), [<length-percentage>](#), ‘[\[solid | dashed | dotted | ... \]](#)’), and a defined behavior on the rendering of the document. Properties values are assigned to various parts of the document via [property declarations](#), which assign the property a value (e.g. ‘[red](#)’, ‘[12pt](#)’, ‘[dotted](#)’) for the associated element or box.

One of the fundamental design principles of CSS is [cascading](#), which allows several style sheets to influence the presentation of a document. When different declarations try to set a value for the same element/property combination, the conflicts must somehow be resolved.

The opposite problem arises when no declarations try to set a the value for an element/property combination. In this case, a value is be found by way of [inheritance](#) or by looking at the property’s [initial value](#).

The [cascading](#) and [defaulting](#) process takes a set of declarations as input, and outputs a [specified value](#) for each property on each element.

The rules for finding the specified value for all properties on all elements in the document are described in this specification. The rules for finding the specified values in the page context and its margin boxes are described in [\[css-page-3\]](#).

§ 1.1. Module Interactions

This section is normative.

This module replaces and extends the rules for assigning property values, cascading, and inheritance defined in [\[CSS2\]](#) chapter 6.

Other CSS modules may expand the definitions of some of the syntax and features defined here. For example, the Media Queries Level 4 specification, when combined with this module, expands the definition of the [<media-query>](#) value type as used in this specification.

For the purpose of this specification, [text nodes](#) are treated as [element](#) children of their associated element, and possess the full set of properties; since they cannot be targeted by selectors all of their computed values are assigned by [defaulting](#).

§ 2. Importing Style Sheets: the ‘@import’ rule

The ‘[@import](#)’ rule allows users to import style rules from other style sheets. If an ‘[@import](#)’ rule refers to a valid stylesheet, user agents must treat the contents of the stylesheet as if they were written in place of the ‘[@import](#)’ rule, with two exceptions:

- If a feature (such as the ‘[@namespace](#)’ rule) *explicitly* defines that it only applies to a particular stylesheet, and not any imported ones, then it doesn’t apply to the imported stylesheet.
- If a feature relies on the relative ordering of two or more constructs in a stylesheet (such as the requirement that ‘[@namespace](#)’ rules must not have any other rules other than ‘[@import](#)’ preceding it), it only applies between constructs in the same stylesheet.

EXAMPLE 1

For example, declarations in style rules from imported stylesheets interact with the cascade as if they were written literally into the stylesheet at the point of the ‘[@import](#)’.

Any ‘[@import](#)’ rules must precede all other valid at-rules and style rules in a style sheet (ignoring ‘[@charset](#)’), or else the ‘[@import](#)’ rule is invalid. The syntax of ‘[@import](#)’ is:

```
@import [ <url> | <string> ]
        [ supports( [ <supports-condition> | <declaration> ] ) ]?
        <media-query-list>? ;
```

where the [<url>](#) or [<string>](#) gives the URL of the style sheet to be imported, and the optional [\[<supports-condition> <declaration>\]](#) and [<media-query-list>](#) (collectively, the *import conditions*) state the conditions under which it applies.

EXAMPLE 2

The following [conditional ‘@import’ rule](#) only loads the style sheet when the UA [supports ‘display: flex’](#), and only applies the style sheet on a [handheld](#) device with a [maximum viewport width](#) of 400px.

```
@import url("narrow.css") supports(display: flex) handheld and (max-width: 400px);
```

If a [<string>](#) is provided, it must be interpreted as a [<url>](#) with the same value.

EXAMPLE 3

The following lines are equivalent in meaning and illustrate both ‘[@import](#)’ syntaxes (one with ‘[url\(\)](#)’ and one with a bare string):

```
@import "mystyle.css";
@import url("mystyle.css");
```

2.1. Conditional ‘[@import](#)’ Rules

The [import conditions](#) allow the import to be media– or feature-support–dependent. In the absence of any import conditions, the import is unconditional. (Specifying ‘[all](#)’ for the [<media-query-list>](#) has the same effect.) If the import conditions do not match, the rules in the imported stylesheet do not apply, exactly as if the imported stylesheet were wrapped in ‘[@media](#)’ and/or ‘[@supports](#)’ blocks with the given conditions.

EXAMPLE 4

The following rules illustrate how ‘[@import](#)’ rules can be made media-dependent:

```
@import url("fingerprint.css") print;
@import url("bluish.css") projection, tv;
@import url("narrow.css") handheld and (max-width: 400px);
```

User agents may therefore avoid fetching a conditional import as long as the [import conditions](#) do not match. Additionally, if a [<supports-condition>](#) blocks the application of the imported style sheet, the UA *must not* fetch the style sheet (unless it is loaded through some other link) and *must* return null for the import rule’s `CSSImportRule.styleSheet` value (even if it is loaded through some other link).

EXAMPLE 5

The following rule illustrates how an author can provide fallback rules for legacy user agents without impacting network performance on newer user agents:

```
@import url("fallback-layout.css") supports(not (display: flex));
@supports (display: flex) {
  ...
}
```

The [import conditions](#) are given by [<media-query-list>](#), which is parsed and interpreted as a [media query list](#), and [<supports-condition>](#), is parsed and interpreted as a [\[\[supports query\]\]](#). If a [<declaration>](#) is given in place of a [<supports-condition>](#), it must be interpreted as a [<supports-decl>](#) (i.e. the extra set of parentheses is implied) and treated as a [<supports-condition>](#).

EXAMPLE 6

For example, the following two lines are equivalent:

```
@import "mystyle.css" supports(display: flex);
@import "mystyle.css" supports((display: flex));
```

The evaluation and full syntax of the [import conditions](#) are defined by the [Media Queries \[MEDIAQ\]](#) and [CSS Conditional Rules \[CSS-CONDITIONAL-3\]](#) specifications.

§ 2.2. Processing Stylesheet Imports

When the same style sheet is imported or linked to a document in multiple places, user agents must process (or act as though they do) each link as though the link were to an independent style sheet.

Note: This does not place any requirements on resource fetching, only how the style sheet is reflected in the CSSOM and used in specs such as this one. Assuming appropriate caching, it is perfectly appropriate for a UA to fetch a style sheet only once, even though it's linked or imported multiple times.

The [cascade origin](#) of an imported style sheet is the cascade origin of the style sheet that imported it.

The [environment encoding](#) of an imported style sheet is the encoding of the style sheet that imported it. [\[css-syntax-3\]](#)

To *fetch an @import*, given an '@import' rule *rule*:

1. Let *parentStylesheet* be *rule*'s [parent CSS style sheet](#). [\[CSSOM\]](#)
2. If *rule* has a [<supports-condition>](#), and that condition is not true, return.
3. Let *parsedUrl* be the result of the [URL parser](#) steps with *rule*'s URL and *parentStylesheet*'s [location](#). If the algorithm returns an error, return. [\[CSSOM\]](#)
4. [Fetch a style resource](#) from *parsedUrl*, with stylesheet *parentStylesheet*, destination "style", CORS mode "no-cors", and processResponse being the following steps given [response](#) *response* and byte stream, null or failure *byteStream*:
 1. If *maybeByteStream* is not a byte stream, return.
 2. If *parentStylesheet* is in [quirks mode](#) and *response* is [CORS-same-origin](#), let *content type* be "text/css". Otherwise, let *content type* be the Content Type metadata of *response*.
 3. If *content type* is not "text/css", return.
 4. Let *importedStylesheet* be the result of [parsing](#) *byteStream* given *parsedUrl*.
 5. Set *importedStylesheet*'s [origin-clean flag](#) to *parentStylesheet*'s origin-clean flag.
 6. If *response* is not [CORS-same-origin](#), unset *importedStylesheet*'s [origin-clean flag](#).
 7. Set *rule*'s [styleSheet](#) to *importedStylesheet*.

§ 2.3. Content-Type of CSS Style Sheets

The processing of imported style sheets depends on the actual type of the linked resource:

- If the resource does not have [Content-Type metadata](#), the type is treated as text/css.
- If the host document is in [quirks mode](#), and the host document's origin is [same origin](#) with the linked resource [response's URL's](#) origin, the type is treated as text/css.
- Otherwise, the type is determined from its [Content-Type metadata](#).

If the linked resource's type is text/css, it must be interpreted as a CSS style sheet. Otherwise, it must be interpreted as a network error.

§ 3. Shorthand Properties

Some properties are *shorthand properties*, meaning that they allow authors to specify the values of several properties with a single property. A [shorthand property](#) sets all of its *longhand sub-properties*, exactly as if expanded in place.

When values are omitted from a [shorthand](#) form, unless otherwise defined, each "missing" [sub-property](#) is assigned its [initial](#)

[value](#).

This means that a [shorthand](#) property declaration always sets *all* of its [sub-properties](#), even those that are not explicitly set. Carelessly used, this might result in inadvertently resetting some sub-properties. Carefully used, a shorthand can guarantee a “blank slate” by resetting sub-properties inadvertently cascaded from other sources.

For example, writing `'background: green'` rather than `'background-color: green'` ensures that the background color overrides any earlier declarations that might have set the background to an image with `'background-image'`.

EXAMPLE 7

For example, the CSS Level 1 `'font'` property is a [shorthand](#) property for setting `'font-style'`, `'font-variant'`, `'font-weight'`, `'font-size'`, `'line-height'`, and `'font-family'` all at once. The multiple declarations of this example:

```
h1 {
  font-weight: bold;
  font-size: 12pt;
  line-height: 14pt;
  font-family: Helvetica;
  font-variant: normal;
  font-style: normal;
}
```

can therefore be rewritten as

```
h1 { font: bold 12pt/14pt Helvetica }
```

As more [‘font’ sub-properties](#) are introduced into CSS, the shorthand declaration resets those to their initial values as well.

In some cases, a [shorthand](#) might have different syntax or special keywords that don’t directly correspond to values of its [sub-properties](#). (In such cases, the shorthand will explicitly define the expansion of its values.)

In other cases, a property might be a *reset-only sub-property* of the shorthand: Like other [sub-properties](#), it is reset to its initial value by the shorthand when unspecified, but the shorthand might not include syntax to set the sub-property to any of its other values. For example, the `'border'` shorthand resets `'border-image'` to its initial value of `'none'`, but has no syntax to set it to anything else. [\[css-backgrounds-3\]](#)

If a [shorthand](#) is specified as one of the [CSS-wide keywords](#) [\[css-values-3\]](#), it sets all of its [sub-properties](#) to that keyword, including any that are [reset-only sub-properties](#). (Note that these keywords cannot be combined with other values in a single declaration, not even in a shorthand.)

Declaring a [shorthand](#) property to be `'!important'` is equivalent to declaring all of its [sub-properties](#) to be `'!important'`.

3.1. Property Aliasing

Properties sometimes change names after being supported for a while, such as vendor-prefixed properties being standardized. The original name still needs to be supported for compatibility reasons, but the new name is preferred. To accomplish this, CSS defines two different ways of “aliasing” old syntax to new syntax.

legacy name aliases

When the old property’s value syntax is identical to that of the new property, the two names are aliased with an operation on par with case-mapping: at parse time, the old property is converted into the new property. This conversion also applies in the CSSOM, both for string arguments and property accessors: requests for the old property name

transparently transfer to the new property name instead.

EXAMPLE 8

For example, if ‘old-name’ is a [legacy name alias](#) for ‘new-name’, `getComputedStyle(e1).oldName` will return the computed style of the newName property, and `e1.style.setPropertyValue("old-name", "value")` will set the ‘new-name’ property to “value”.

legacy shorthands

When the old property has a distinct syntax from the new property, the two names are aliased using the [shorthand](#) mechanism. These shorthands are defined to be [legacy shorthands](#), and their use is *deprecated*. They otherwise behave exactly as regular shorthands, except that the CSSOM will not use them when serializing declarations. [\[CSSOM\]](#)

EXAMPLE 9

For example, the ‘page-break-*’ properties are [legacy shorthands](#) for the ‘break-*’ properties (see [CSS Fragmentation 3 § 3.4 Page Break Aliases: the page-break-before, page-break-after, and page-break-inside properties](#)).

Setting ‘page-break-before: always’ expands to ‘break-before: page’ at parse time, like other shorthands do. Similarly, if ‘break-before: page’ is set, calling `getComputedStyle(e1).pageBreakBefore` will return “always”. However, when serializing a style block (see [CSSOM 1 § 6.7.2 Serializing CSS Values](#)), the ‘page-break-before’ property will never be chosen as the shorthand to serialize to, regardless of whether it or ‘break-before’ was specified; instead, ‘break-before’ will always be chosen.

3.2. Resetting All Properties: the ‘all’ property

<i>Name:</i>	‘all’
<i>Value:</i>	initial inherit unset revert
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	see individual properties
<i>Inherited:</i>	see individual properties
<i>Percentages:</i>	see individual properties
<i>Computed value:</i>	see individual properties
<i>Animation type:</i>	see individual properties
<i>Canonical order:</i>	per grammar

The ‘all’ property is a [shorthand](#) that resets *all* CSS properties except ‘[direction](#)’ and ‘[unicode-bidi](#)’. It only accepts the [CSS-wide keywords](#). It does not reset [custom properties](#) [\[css-variables-1\]](#).

Note: The excepted CSS properties ‘[direction](#)’ and ‘[unicode-bidi](#)’ are actually markup-level features, and [should not be set in the author’s style sheet](#). (They exist as CSS properties only to style document languages not supported by the UA.) Authors should use the appropriate markup, such as HTML’s `dir` attribute, instead. [\[css-writing-modes-3\]](#)

EXAMPLE 10

For example, if an author specifies ‘[all: initial](#)’ on an element, it will block all inheritance and reset all properties, as if no rules appeared in the author, user, or user-agent levels of the cascade.

This can be useful for the root element of a "widget" included in a page, which does not wish to inherit the styles of the outer page. Note, however, that any "default" style applied to that element (such as, e.g. ‘[display: block](#)’ from the UA style sheet on block elements such as `<div>`) will also be blown away.

§ 4. Value Processing

Once a user agent has parsed a document and constructed a document tree, it must assign, to every element in the [flat tree](#), and correspondingly to every box in the formatting structure, a value to every property that applies to the target media type.

The final value of a CSS property for a given element or box is the result of a multi-step calculation:

1. First, all the [declared values](#) applied to an element are collected, for each property on each element. There may be zero or many declared values applied to the element.
2. Cascading yields the [cascaded value](#). There is at most one cascaded value per property per element.
3. Defaulting yields the [specified value](#). Every element has exactly one specified value per property.
4. Resolving value dependencies yields the [computed value](#). Every element has exactly one computed value per property.
5. Formatting the document yields the [used value](#). An element only has a used value for a given property if that property applies to the element.
6. Finally, the used value is transformed to the [actual value](#) based on constraints of the display environment. As with the [used value](#), there may or may not be an actual value for a given property on an element.

Elements that are not [connected](#) or are not part of the document’s [flattened element tree](#) do not participate in CSS value processing, and do not have [declared](#), [cascaded](#), [specified](#), [computed](#), [used](#), or [actual](#) values, even if they potentially have style declarations assigned to them (for example, by a `style` attribute).

§ 4.1. Declared Values

Each property declaration [applied to an element](#) contributes a ***declared value*** for that property associated with the element. See [Filtering Declarations](#) for details.

These values are then processed by the [cascade](#) to choose a single “winning value”.

§ 4.1.1. Value Aliasing

Some property values have ***legacy value aliases***: at parse time, the legacy syntax is converted into the new syntax, resulting in a [declared value](#) different from the parsed input. These aliases are typically used for handling legacy compatibility requirements, such as converting [vendor-prefixed](#) values to their standard equivalents.

§ 4.2. Cascaded Values

The ***cascaded value*** represents the result of [the cascade](#): it is the [declared value](#) that wins the cascade (is sorted first in the [output of the cascade](#)). If the output of the cascade is an empty list, there is no [cascaded value](#).

§ 4.3. Specified Values

The **specified value** is the value of a given property that the style sheet authors intended for that element. It is the result of putting the [cascaded value](#) through the [defaulting](#) processes, guaranteeing that a [specified value](#) exists for every property on every element.

In many cases, the [specified value](#) is the [cascaded value](#). However, if there is no cascaded value at all, the specified value is [defaulted](#). The [CSS-wide keywords](#) are handled specially when they are the cascaded value of a property, setting the specified value as required by that keyword, see [§ 7.3 Explicit Defaulting](#).

§ 4.4. Computed Values

The **computed value** is the result of resolving the [specified value](#) as defined in the “Computed Value” line of the property definition table, generally absolutizing it in preparation for [inheritance](#).

Note: The [computed value](#) is the value that is transferred from parent to child during [inheritance](#). For historical reasons, it is not necessarily the value returned by the [getComputedStyle\(\)](#) function, which sometimes returns [used values](#). [\[CSSOM\]](#) Furthermore, the computed value is an abstract data representation: their definitions reflect that data representation, not how that data is serialized. For example, serialization rules often allow omitting certain values which are implied during parsing; but those values are nonetheless part of the computed value.

EXAMPLE 11

A [specified value](#) can be either absolute (i.e., not relative to another value, as in ‘[red](#)’ or ‘[2mm](#)’) or relative (i.e., relative to another value, as in ‘[auto](#)’, ‘[2em](#)’). Computing a relative value generally absolutizes it:

- values with relative units (‘[em](#)’, ‘[ex](#)’, ‘[vh](#)’, ‘[vw](#)’) must be made absolute by multiplying with the appropriate reference size
- certain keywords (e.g., ‘[smaller](#)’, ‘[bolder](#)’) must be replaced according to their definitions
- percentages on some properties must be multiplied by a reference value (defined by the property)
- valid relative URLs must be resolved to become absolute.

See examples (f), (g) and (h) in the [table below](#).

Note: In general, the [computed value](#) resolves the [specified value](#) as far as possible without laying out the document or performing other expensive or hard-to-parallelize operations, such as resolving network requests or retrieving values other than from the element and its parent.

The [computed value](#) exists even when the property does not apply. However, some properties may change how they determine the computed value based on whether the property [applies to](#) the element.

§ 4.5. Used Values

The **used value** is the result of taking the [computed value](#) and completing any remaining calculations to make it the absolute theoretical value used in the formatting of the document.

EXAMPLE 12

For example, a declaration of ‘[width: auto](#)’ can’t be resolved into a length without knowing the layout of the element’s ancestors, so the [computed value](#) is ‘[auto](#)’, while the [used value](#) is an absolute length, such as ‘[100px](#)’. [\[CSS2\]](#)

EXAMPLE 13

As another example, a `<div>` might have a computed `‘break-before’` value of `‘auto’`, but acquire a used `‘break-before’` value of `‘page’` by propagation from its first child. [\[css-break-3\]](#)

If a property does not [apply to](#) this element or box type then it has no [used value](#) for that property.

EXAMPLE 14

For example, the `‘flex’` property has no [used value](#) on elements that aren’t [flex items](#).

4.5.1. Applicable Properties

If a property does not **apply to** an element or box type—as noted in its “Applies to” line—this means it does not directly take effect on that type of box or element.

Note: A property that does not apply can still have *indirect* formatting effects if its computed value affects the computation of other properties that do apply; and of course its [computed value](#), which always exists, can still inherit to descendants and take effect on them.

EXAMPLE 15

Even though `‘writing-mode’` and `‘text-orientation’` do not apply to table rows (they do not affect how the table row or its children are laid out), setting them on such boxes will still affect the calculation of font relative units such as `‘ch’`, and thus possibly any property that takes a `<length>`.

EXAMPLE 16

Setting `‘text-transform’` on an HTML `<p>` element (which is `‘display: block’` by default) will have an effect, even though `‘text-transform’` only applies to [inline boxes](#), because the property inherits into the paragraph’s anonymous [root inline box](#) and applies to the text it contains.

Note: A property defined to apply to “all elements” applies to all elements and [display types](#), but not necessarily to all [pseudo-element](#) types, since pseudo-elements often have their own specific rendering models or other restrictions. The `‘::before’` and `‘::after’` pseudo-elements, however, are defined to generate boxes almost exactly like normal elements and are therefore defined accept all properties that apply to “all elements”. See [\[CSS-PSEUDO-4\]](#) for more information about pseudo-elements.

4.6. Actual Values

A [used value](#) is in principle ready to be used, but a user agent may not be able to make use of the value in a given environment. For example, a user agent may only be able to render borders with integer pixel widths and may therefore have to approximate the [used](#) width. Also, the font size of an element may need adjustment based on the availability of fonts or the value of the `‘font-size-adjust’` property. The **actual value** is the used value after any such adjustments have been made.

Note: By probing the actual values of elements, much can be learned about how the document is laid out. However, not all information is recorded in the actual values. For example, the actual value of the `‘page-break-after’` property does not reflect whether there is a page break or not after the element. Similarly, the actual value of `‘orphans’` does not reflect how many orphan lines there is in a certain element. See examples (j) and (k) in the [table below](#).

4.7. Examples

Examples of CSS Value Computation

	Property	Winning declaration	Cascaded value	Specified value	Computed value	Used value	Actual value
(a)	<u>‘text-align’</u>	text-align: left	‘left’	‘left’	‘left’	‘left’	‘left’
(b)	<u>‘border-top-width’</u> , <u>‘border-right-width’</u> , <u>‘border-bottom-width’</u> , <u>‘border-left-width’</u>	border-width: inherit	‘inherit’	‘4.2px’	‘4.2px’	‘4.2px’	‘4px’
(c)	<u>‘width’</u>	(none)	(none)	‘auto’ (initial value)	‘auto’	‘120px’	‘120px’
(d)	<u>‘list-style-position’</u>	list-style-position: inherit	‘inherit’	‘inside’	‘inside’	‘inside’	‘inside’
(e)	<u>‘list-style-position’</u>	list-style-position: initial	‘initial’	‘outside’ (initial value)	‘outside’	‘outside’	‘outside’
(f)	<u>‘font-size’</u>	font-size: 1.2em	‘1.2em’	‘1.2em’	‘14.1px’	‘14.1px’	‘14px’
(g)	<u>‘width’</u>	width: 80%	‘80%’	‘80%’	‘80%’	‘354.2px’	‘354px’
(h)	<u>‘width’</u>	width: auto	‘auto’	‘auto’	‘auto’	‘134px’	‘134px’
(i)	<u>‘height’</u>	height: auto	‘auto’	‘auto’	‘auto’	‘176px’	‘176px’
(j)	<u>‘page-break-after’</u>	(none)	(none)	‘auto’ (initial value)	‘auto’	‘auto’	‘auto’
(k)	<u>‘orphans’</u>	orphans: 3	‘3’	‘3’	‘3’	‘3’	‘3’

5. Filtering

In order to find the [declared values](#), implementations must first identify all declarations that apply to each element. A declaration applies to an element if:

- It belongs to a style sheet that currently applies to this document.
- It is not qualified by a conditional rule [\[CSS-CONDITIONAL-3\]](#) with a false condition.

- It belongs to a style rule whose selector matches the element. [\[SELECT\]](#) (Taking [scoping](#) into account, if necessary.)
- It is syntactically valid: the declaration's property is a known property name, and the declaration's value matches the syntax for that property.

The values of the declarations that apply form, for each property on each element, a list of [declared values](#). The next section, the [cascade](#), prioritizes these lists.

§ 6. Cascading

The ***cascade*** takes an unordered list of [declared values](#) for a given property on a given element, sorts them by their declaration's precedence as determined below, and outputs a single [cascaded value](#).

§ 6.1. Cascade Sorting Order

The cascade sorts declarations according to the following criteria, in descending order of priority:

¶ Origin and Importance

The [origin](#) of a declaration is based on where it comes from and its [importance](#) is whether or not it is declared with `'!important'` (see [below](#)). The precedence of the various origins is, in descending order:

1. Transition declarations [\[css-transitions-1\]](#)
2. [Important user agent](#) declarations
3. [Important user](#) declarations
4. [Important author](#) declarations
5. Animation declarations [\[css-animations-1\]](#)
6. [Normal author](#) declarations
7. [Normal user](#) declarations
8. [Normal user agent](#) declarations

Declarations from [origins](#) earlier in this list win over declarations from later origins.

¶ Context

A document language can provide for blending declarations sourced from different ***encapsulation contexts***, such as the nested [tree contexts](#) of [shadow trees](#) in the [\[DOM\]](#).

When comparing two declarations that are sourced from different [encapsulation contexts](#), then for [normal](#) rules the declaration from the outer context wins, and for [important](#) rules the declaration from the inner context wins. For this purpose, [\[DOM\] tree contexts](#) are considered to be nested in [shadow-including tree order](#).

Note: This effectively means that [normal](#) declarations belonging to an [encapsulation context](#) can set defaults that are easily overridden by the outer context, while [important](#) declarations belonging to an encapsulation context can enforce requirements that cannot be overridden by the outer context.

¶ Specificity

The [Selectors module](#) [\[SELECT\]](#) describes how to compute the specificity of a selector. Each declaration has the same specificity as the style rule it appears in. For the purpose of this step, declarations that do not belong to a style rule (such as the [contents of a style attribute](#)) are considered to have a specificity higher than any selector. The declaration with the highest specificity wins.

¶ Order of Appearance

The last declaration in document order wins. For this purpose:

- Declarations from ['imported style sheets'](#) are ordered as if their style sheets were substituted in place of the

‘@import’ rule.

- Declarations from style sheets independently linked by the originating document are treated as if they were concatenated in linking order, as determined by the host document language.
- Declarations from style attributes are ordered according to the document order of the element the style attribute appears on, and are all placed after any style sheets.

The *output of the cascade* is a (potentially empty) sorted list of [declared values](#) for each property on each element.

6.2. Cascading Origins

Each style rule has a *cascade origin*, which determines where it enters the cascade. CSS defines three core [origins](#):

Author Origin

The author specifies style sheets for a source document according to the conventions of the document language. For instance, in HTML, style sheets may be included in the document or linked externally.

User Origin

The user may be able to specify style information for a particular document. For example, the user may specify a file that contains a style sheet or the user agent may provide an interface that generates a user style sheet (or behaves as if it did).

User-Agent Origin

Conforming user agents must apply a default style sheet (or behave as if they did). A user agent’s default style sheet should present the elements of the document language in ways that satisfy general presentation expectations for the document language (e.g., for visual browsers, the EM element in HTML is presented using an italic font). See e.g. the [HTML user agent style sheet](#). [\[HTML\]](#)

Extensions to CSS define the following additional [origins](#):

Animation Origin

CSS Animations [\[css-animations-1\]](#) generate “virtual” rules representing their effects when running.

Transition Origin

Like CSS Animations, CSS Transitions [\[css-transitions-1\]](#) generate “virtual” rules representing their effects when running.

6.3. Important Declarations: the ‘!important’ annotation

CSS attempts to create a balance of power between author and user style sheets. By default, rules in an author’s style sheet override those in a user’s style sheet, which override those in the user-agent’s default style sheet. To balance this, a declaration can be marked [important](#), which increases its weight in the cascade and inverts the order of precedence.

A declaration is *important* if it has a ‘!important’ annotation as defined by [\[css-syntax-3\]](#), i.e. if the last two (non-whitespace, non-comment) tokens in its value are the delimiter token ‘!’ followed by the identifier token ‘important’. All other declarations are *normal* (non-[important](#)).

EXAMPLE 17

```
[hidden] { display: none !important; }
```

An [important](#) declaration takes precedence over a [normal](#) declaration. Author and user style sheets may contain important declarations, with [user-origin](#) important declarations overriding [author-origin](#) important declarations. This CSS feature improves accessibility of documents by giving users with special requirements (large fonts, color combinations, etc.) control over presentation.

[Important](#) declarations from all origins take precedence over animations. This allows authors to override animated values in important cases. (Animated values normally override all other rules.) [\[css-animations-1\]](#)

[User-agent style sheets](#) may also contain [important](#) declarations. These override all [author](#) and [user](#) declarations.

EXAMPLE 18

The first rule in the user’s style sheet in the following example contains an ‘[!important](#)’ declaration, which overrides the corresponding declaration in the author’s style sheet. The declaration in the second rule will also win due to being marked ‘[!important](#)’. However, the third declaration in the user’s style sheet is not ‘[!important](#)’ and will therefore lose to the second rule in the author’s style sheet (which happens to set style on a [shorthand](#) property). Also, the third author rule will lose to the second author rule since the second declaration is ‘[!important](#)’. This shows that ‘[!important](#)’ declarations have a function also within author style sheets.

```
/* From the user’s style sheet */
p { text-indent: 1em !important }
p { font-style: italic !important }
p { font-size: 18pt }

/* From the author’s style sheet */
p { text-indent: 1.5em !important }
p { font: normal 12pt sans-serif !important }
p { font-size: 24pt }
```

Property	Winning value
‘text-indent’	‘1em’
‘font-style’	‘italic’
‘font-size’	‘12pt’
‘font-family’	‘sans-serif’

§ 6.4. Precedence of Non-CSS Presentational Hints

The UA may choose to honor presentational hints in a source document’s markup, for example the `bgcolor` attribute or [<S>](#) element in [\[HTML\]](#). All document language-based styling must be translated to corresponding CSS rules and enter the cascade as rules in either the [UA-origin](#) or a special-purpose *author presentational hint origin* between the regular [user origin](#) and the [author origin](#). For the purpose of [cascading](#) this [author presentational hint origin](#) is treated as an independent [origin](#), but for the purpose of the ‘[revert](#)’ keyword it is considered part of the author origin.

A document language may define whether such a presentational hint enters the [cascade](#) as [UA-origin](#) or [author-origin](#); if so, the UA must behave accordingly. For example, [\[SVG11\]](#) maps its presentation attributes into the author origin.

Note: Presentational hints entering the [cascade](#) as [UA-origin](#) rules can be overridden by [author-origin](#) or [user-origin](#) styles. Presentational hints entering the cascade as [author presentational hint origin](#) rules can be overridden by author-origin styles, but not by non-[important](#) user-origin styles. Host languages should choose the appropriate origin for presentational hints with these considerations in mind.

§ 7. Defaulting

When the [cascade](#) does not result in a value, the [specified value](#) must be found some other way. [Inherited properties](#) draw their defaults from their parent element through [inheritance](#); all other properties take their [initial value](#). Authors can explicitly request inheritance or initialization via the [‘inherit’](#) and [‘initial’](#) keywords.

7.1. Initial Values

Each property has an *initial value*, defined in the property’s definition table. If the property is not an [inherited property](#), and the [cascade](#) does not result in a value, then the [specified value](#) of the property is its [initial value](#).

7.2. Inheritance

Inheritance propagates property values from parent elements to their children. The *inherited value* of a property on an element is the [computed value](#) of the property on the element’s parent element. For the root element, which has no parent element, the [inherited value](#) is the [initial value](#) of the property.

For a [\[DOM\]](#) tree with shadows, inheritance operates on the [flattened element tree](#). This means that slotted elements inherit from the [<slot>](#) they’re assigned to, rather than directly from their [light tree](#) parent. [Pseudo-elements](#) inherit according to the fictional tag sequence described for each pseudo-element. [\[CSS-PSEUDO-4\]](#)

Some properties are *inherited properties*, as defined in their property definition table. This means that, unless the [cascade](#) results in a value, the value will be determined by [inheritance](#).

A property can also be explicitly inherited. See the [‘inherit’](#) keyword.

Note: Inheritance follows the document tree and is not intercepted by [anonymous boxes](#), or otherwise affected by manipulations of the box tree.

7.3. Explicit Defaulting

Several CSS-wide property values are defined below; declaring a property to have these values explicitly specifies a particular defaulting behavior. As specified in [CSS Values and Units \[css-values-3\]](#), all CSS properties can accept these values.

7.3.1. Resetting a Property: the [‘initial’](#) keyword

If the [cascaded value](#) of a property is the *‘initial’* keyword, the property’s [specified value](#) is its [initial value](#).

7.3.2. Explicit Inheritance: the [‘inherit’](#) keyword

If the [cascaded value](#) of a property is the *‘inherit’* keyword, the property’s [specified](#) and [computed values](#) are the [inherited value](#).

7.3.3. Erasing All Declarations: the [‘unset’](#) keyword

If the [cascaded value](#) of a property is the *‘unset’* keyword, then if it is an inherited property, this is treated as [‘inherit’](#), and if it is not, this is treated as [‘initial’](#). This keyword effectively erases all [declared values](#) occurring earlier in the [cascade](#), correctly inheriting or not as appropriate for the property (or all longhands of a [shorthand](#)).

7.3.4. Rolling Back Cascade Origins: the ‘[revert](#)’ keyword

If the [cascaded value](#) of a property is the ‘[revert](#)’ keyword, the behavior depends on the [cascade origin](#) to which the declaration belongs:

[user-agent origin](#)

Equivalent to ‘[unset](#)’.

[user origin](#)

Rolls back the [cascaded value](#) to the user-agent level, so that the [specified value](#) is calculated as if no [author-origin](#) or [user-origin](#) rules were specified for this property on this element.

[author origin](#)

Rolls back the [cascaded value](#) to the user level, so that the [specified value](#) is calculated as if no [author-origin](#) rules were specified for this property on this element. For the purpose of ‘[revert](#)’, this origin includes the Animation [origin](#).

8. Changes

8.1. Changes since the 15 Oct 2021 Working Draft

Non-trivial changes since the [15 October 2021 Working Draft](#):

- Updated [@import](#) grammar for media queries and supports conditions
- Allowed functional notation parse-time aliases
- Defined fetching an [@import](#), in terms of Fetch
- Added [§ 4.1.1 Value Aliasing](#) section. ([Issue 6193](#))

8.2. Changes Since the 28 August 2018 Candidate Recommendation

Non-trivial changes since the [19 March 2021 Working Draft](#) include:

- Defined [author presentational hint origin](#) to handle author-origin presentational hints, instead of relying on zero-specificity and source order, to correctly define their interaction with the [encapsulation context](#) aspect of the cascade. ([Issue 66749](#))

Non-trivial changes since the [18 August 2020 Working Draft](#) include:

- Removed possibility of [legacy name aliases](#) to map subsets of the value space, since they are simple name aliases. ([Issue 4839](#))
- Gave concept of [applies to](#) its own section and add some notes about implications. (Issues [1861](#) and [5565](#))
- Defined the term [property](#). ([Issue 5633](#))
- Defined value processing of elements that are not part of the tree. (Issue [1964](#) and [1548](#))

[Elements that are not connected or are not part of the document’s flattened element tree do not participate in CSS value processing, and do not have declared, cascaded, specified, computed, used, or actual values, even if they potentially have style declarations assigned to them \(for example, by a style attribute\).](#)

- Clarify origin comparison for quirks mode Content-Type assumptions in [§ 2.3 Content-Type of CSS Style Sheets](#). ([Issue 4838](#))

Non-trivial changes since the [28 August 2018 Candidate Recommendation](#) include:

- Added [context](#) to the [cascade](#) sort criteria to accommodate Shadow DOM. [\[DOM\]](#) ([Issue 5372](#))

- Defined that, in consideration of [shadow trees](#), [inheritance](#) operates over the [flattened element tree](#).
- Removed scoping from the [cascade](#) sort criteria, because it has not been implemented.

§ 8.3. Changes Since the 14 January 2016 Candidate Recommendation

Non-trivial changes since the [14 January 2016 Working Draft](#) include:

- Precisely defined the types of aliasing that CSS uses. ([Issue 866](#)) See [§ 3.1 Property Aliasing](#).
- Clarified that [‘revert’](#) only affects the cascaded value, not the inherited value.

user origin

Rolls back the [cascade cascaded value to the user-agent level](#), so that the [specified value is calculated as if no author-level or user-level rules were specified for this property on this element](#).

author origin

Rolls back the [cascade cascaded value](#) to the user level, so that the [specified value](#) is calculated as if no author-level rules were specified for this property [on this element](#).

- Clarified that [custom properties](#) are not reset by the [‘all’](#) shorthand. ([2518](#))

The [‘all’](#) property is a [shorthand](#) that resets *all* CSS properties except [‘direction’](#) and [‘unicode-bidi’](#). ... [It does not reset custom properties \[css-variables-1\]](#).

- Defined more precisely that imported stylesheets are interpreted separately from the importing stylesheet, in terms of ordering of rules, etc.

If an [‘@import’](#) rule refers to a valid stylesheet, user agents must treat the contents of the stylesheet as if they were written in place of the [‘@import’](#) rule, [with two exceptions](#):

- [If a feature \(such as the ‘@namespace’ rule\) explicitly defines that it only applies to a particular stylesheet, and not any imported ones, then it doesn’t apply to the imported stylesheet.](#)
- [If a feature relies on the relative ordering of two or more constructs in a stylesheet \(such as the requirement that ‘@charset’ must not have any other content preceding it\), it only applies between constructs in the same stylesheet.](#)

- Specified that text nodes are considered children of their parent element, and receive styles via defaulting, as their properties are now observable distinct from their inline parent’s via [‘display: contents’ \[css-display-3\]](#).

[For the purpose of this specification, text nodes are treated as element children of their associated element, and possess the full set of properties; since they cannot be targeted by selectors all of their computed values are assigned by defaulting.](#)

- Removed any mention of the obsolete “override” origin, originally defined by [DOM Level 2 Style](#) and later abandoned. ([Issue 1385](#))

A [Disposition of Comments](#) is available.

§ 8.4. Changes Since the 21 April 2015 Working Draft

Changes since the [21 April 2015 Working Draft](#) include:

- Renamed ‘[default](#)’ keyword to ‘[revert](#)’.
- Allowed dropping duplicate parentheses in ‘[supports\(\)](#)’ syntax when it only contains one declaration.

§ 8.5. Additions Since Level 3

The following features have been added since [Level 3](#):

- Introduced ‘[revert](#)’ keyword, for rolling back the cascade.
- Introduced ‘[supports\(\)](#)’ syntax for supports-conditional ‘[@import](#)’ rules.
- Added [encapsulation context](#) to the [cascade](#) sort criteria to accommodate Shadow DOM. [\[DOM\]](#)
- Defined the property two aliasing mechanisms CSS uses to support legacy syntaxes. See [§ 3.1 Property Aliasing](#).

§ 8.6. Additions Since Level 2

The following features have been added since [Level 2](#):

- The ‘[all](#)’ shorthand
- The ‘[initial](#)’ keyword
- The ‘[unset](#)’ keyword
- Incorporation of animations and transitions into the [cascade](#).

§ Acknowledgments

David Baron, Tantek Çelik, Simon Sapin, Noam Rosenthal, and Boris Zbarsky contributed to this specification.

§ Privacy and Security Considerations

- The cascade process does not distinguish between same-origin and cross-origin stylesheets, enabling the content of cross-origin stylesheets to be inferred from the computed styles they apply to a document.
- User preferences and UA defaults expressed via application of style rules are exposed by the cascade process, and can be inferred from the computed styles they apply to a document.
- The ‘[@import](#)’ rule does not apply the [CORS protocol](#) to loading cross-origin stylesheets, instead allowing them to be freely imported and applied.
- The ‘[@import](#)’ rule assumes that resources without [Content-Type metadata](#) (or any same-origin file if the host document is in quirks mode) are `text/css`, potentially allowing arbitrary files to be imported into the page and interpreted as CSS, potentially allowing sensitive data to be inferred from the computed styles they apply to a document.

§ 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 19

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 unprefix 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 unprefix 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.

§ CR exit criteria

For this specification to be advanced to Proposed Recommendation, there must be at least two independent, interoperable implementations of each feature. Each feature may be implemented by a different set of products, there is no requirement that all features be implemented by a single product. For the purposes of this criterion, we define the following terms:

independent

each implementation must be developed by a different party and cannot share, reuse, or derive from code used by another qualifying implementation. Sections of code that have no bearing on the implementation of this specification are exempt from this requirement.

interoperable

passing the respective test case(s) in the official CSS test suite, or, if the implementation is not a Web browser, an equivalent test. Every relevant test in the test suite should have an equivalent test created if such a user agent (UA) is to be used to claim interoperability. In addition if such a UA is to be used to claim interoperability, then there must one or more additional UAs which can also pass those equivalent tests in the same way for the purpose of interoperability. The equivalent tests must be made publicly available for the purposes of peer review.

implementation

a user agent which:

1. implements the specification.
2. is available to the general public. The implementation may be a shipping product or other publicly available version (i.e., beta version, preview release, or "nightly build"). Non-shipping product releases must have implemented the feature(s) for a period of at least one month in order to demonstrate stability.
3. is not experimental (i.e., a version specifically designed to pass the test suite and is not intended for normal usage going forward).

The specification will remain Candidate Recommendation for at least six months.

§ Index

§ Terms defined by this specification

[actual](#), in § 4.6

[actual value](#), in § 4.6

[all](#), in § 3.2

[Animation Origin](#), in § 6.2[apply to](#), in § 4.5.1[author origin](#), in § 6.2[author-origin](#), in § 6.2[author presentational hint origin](#), in § 6.4[author style sheet](#), in § 6.2[cascade](#), in § 6[cascaded](#), in § 4.2[cascaded value](#), in § 4.2[cascade origin](#), in § 6.2[computed](#), in § 4.4[computed value](#), in § 4.4[context](#), in § 6.1[declared](#), in § 4.1[declared value](#), in § 4.1[encapsulation contexts](#), in § 6.1[fetch an @import](#), in § 2.2[@import](#), in § 2[importance](#), in § 6.3[important](#), in § 6.3[import conditions](#), in § 2[inherit](#)[definition of](#), in § 7.2[value for all](#), in § 7.3.2[inheritance](#), in § 7.2[inherited property](#), in § 7.2[inherited value](#), in § 7.2[initial](#), in § 7.3.1[initial value](#), in § 7.1[legacy name alias](#), in § 3.1[legacy shorthand](#), in § 3.1[legacy value alias](#), in § 4.1.1[longhand](#), in § 3[longhand property](#), in § 3[normal](#), in § 6.3[origin](#), in § 6.2[output of the cascade](#), in § 6.1[property](#), in § 1[reset-only sub-property](#), in § 3[revert](#), in § 7.3.4[shorthand](#), in § 3[shorthand property](#), in § 3[specified](#), in § 4.3[specified value](#), in § 4.3[sub-property](#), in § 3[Transition Origin](#), in § 6.2[UA origin](#), in § 6.2[UA-origin](#), in § 6.2[UA style sheet](#), in § 6.2[unset](#), in § 7.3.3[used](#), in § 4.5[used value](#), in § 4.5[user-agent origin](#), in § 6.2[user-agent style sheet](#), in § 6.2[user origin](#), in § 6.2[user-origin](#), in § 6.2[user style sheet](#), in § 6.2

§ Terms defined by reference

[css-2021] defines the following terms:

vendor-prefixed

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

auto

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

background

background-color

background-image

border

border-bottom-width

border-image

border-left-width

border-right-width

border-style

border-top-width

dotted

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

break-before

orphans

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

page

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

<color>

color

red

[CSS-CONDITIONAL-3] defines the following terms:

<supports-condition>

<supports-decl>

@media

@supports

[css-conditional-5] defines the following terms:

supports()

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

display type

element

inline box

text node

[css-flexbox-1] defines the following terms:

flex

flex item

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

bolder

font

font-family

font-style

font-variant

font-weight

italic

sans-serif

[css-fonts-5] defines the following terms:

font-size

font-size-adjust

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

root inline box

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

list-style-position

[CSS-PSEUDO-4] defines the following terms:

::after

::before

[css-scoping-1] defines the following terms:

flat tree

flattened element tree

shadow tree

tree context

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

height

width

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

@charset

environment encoding

parse a stylesheet

property declarations

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

text-align

text-indent

text-transform

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

ex

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

<length-percentage>

<length>

<string>

<url>

?

ch

css-wide keywords

em

fetch a style resource

url()

vh

vw

|

[css-variables-1] defines the following terms:

custom property

[css-writing-modes-3] defines the following terms:

direction

unicode-bidi

[css-writing-modes-4] defines the following terms:

text-orientation

writing-mode

[CSS2] defines the following terms:

display

line-height

page-break-after

page-break-before

[CSSOM] defines the following terms:

getComputedStyle(elt)

location

origin-clean flag

parent css style sheet

styleSheet

[DOM] defines the following terms:

connected

light tree

quirks mode

shadow-including tree order

[FETCH] defines the following terms:

cors protocol

response

url

[HTML] defines the following terms:

content-type metadata

cors-same-origin

p

s

same origin

slot

[MEDIAQ] defines the following terms:

<media-query-list>

<media-query>

all

media query list

[selectors-4] defines the following terms:

pseudo-element

[URL] defines the following terms:

url parser

§ References

§ Normative References

[CSS-2021]

Tab Atkins Jr.; Erika Etemad; Florian Rivoal. *CSS Snapshot 2021*. 31 December 2021. NOTE. URL: <https://www.w3.org/TR/css-2021/>

[CSS-ANIMATIONS-1]

Dean Jackson; et al. *CSS Animations Level 1*. 11 October 2018. WD. URL: <https://www.w3.org/TR/css-animations-1/>

[CSS-BACKGROUNDS-3]

Bert Bos; Erika Etemad; Brad Kemper. *CSS Backgrounds and Borders Module Level 3*. 26 July 2021. CR. URL: <https://www.w3.org/TR/css-backgrounds-3/>

[CSS-COLOR-4]

Tab Atkins Jr.; Chris Lilley; Lea Verou. *CSS Color Module Level 4*. 15 December 2021. WD. URL: <https://www.w3.org/TR/css-color-4/>

[CSS-CONDITIONAL-3]

David Baron; Erika Etemad; Chris Lilley. *CSS Conditional Rules Module Level 3*. 23 December 2021. CR. URL: <https://www.w3.org/TR/css-conditional-3/>

[CSS-CONDITIONAL-5]

David Baron; Erika Etemad; Chris Lilley. *CSS Conditional Rules Module Level 5*. 21 December 2021. WD. URL: <https://www.w3.org/TR/css-conditional-5/>

[CSS-DISPLAY-3]

Tab Atkins Jr.; Erika Etemad. *CSS Display Module Level 3*. 3 September 2021. CR. URL: <https://www.w3.org/TR/css-display-3/>

[CSS-FONTS-5]

Myles Maxfield; Chris Lilley. *CSS Fonts Module Level 5*. 21 December 2021. WD. URL: <https://www.w3.org/TR/css-fonts-5/>

[CSS-PSEUDO-4]

Daniel Glazman; Erika Etemad; Alan Stearns. *CSS Pseudo-Elements Module Level 4*. 31 December 2020. WD. URL: <https://www.w3.org/TR/css-pseudo-4/>

[CSS-SCOPING-1]

Tab Atkins Jr.; Erika Etemad. *CSS Scoping Module Level 1*. 3 April 2014. WD. URL: <https://www.w3.org/TR/css-scoping-1/>

[CSS-SYNTAX-3]

Tab Atkins Jr.; Simon Sapin. *CSS Syntax Module Level 3*. 24 December 2021. CR. URL: <https://www.w3.org/TR/css-syntax-3/>

[CSS-TRANSITIONS-1]

David Baron; et al. *CSS Transitions*. 11 October 2018. WD. URL: <https://www.w3.org/TR/css-transitions-1/>

[CSS-VALUES-3]

Tab Atkins Jr.; Erika Etemad. *CSS Values and Units Module Level 3*. 6 June 2019. CR. URL: <https://www.w3.org/TR/css-values-3/>

[CSS-VALUES-4]

Tab Atkins Jr.; Erika Etemad. *CSS Values and Units Module Level 4*. 16 December 2021. WD. URL: <https://www.w3.org/TR/css-values-4/>

[CSS-VARIABLES-1]

Tab Atkins Jr.. *CSS Custom Properties for Cascading Variables Module Level 1*. 11 November 2021. CR. URL: <https://www.w3.org/TR/css-variables-1/>

[CSS-WRITING-MODES-3]

Erika Etemad; Koji Ishii. *CSS Writing Modes Level 3*. 10 December 2019. REC. URL: <https://www.w3.org/TR/css-writing-modes-3/>

[CSS2]

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]

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

[DOM]

Anne van Kesteren. *DOM Standard*. Living Standard. URL: <https://dom.spec.whatwg.org/>

[FETCH]

Anne van Kesteren. *Fetch Standard*. Living Standard. URL: <https://fetch.spec.whatwg.org/>

[HTML]

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

[MEDIAQ]

Florian Rivoal; Tab Atkins Jr.. *Media Queries Level 4*. 25 December 2021. CR. URL: <https://www.w3.org/TR/mediaqueries-4/>

[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>

[SELECT]

Tantek Çelik; et al. *Selectors Level 3*. 6 November 2018. REC. URL: <https://www.w3.org/TR/selectors-3/>

[SELECTORS-4]

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

[URL]

Anne van Kesteren. *URL Standard*. Living Standard. URL: <https://url.spec.whatwg.org/>

§ Informative References

[CSS-ALIGN-3]

Elika Etemad; Tab Atkins Jr.. *CSS Box Alignment Module Level 3*. 24 December 2021. WD. URL: <https://www.w3.org/TR/css-align-3/>

[CSS-BREAK-3]

Rossen Atanassov; Elika Etemad. *CSS Fragmentation Module Level 3*. 4 December 2018. CR. URL: <https://www.w3.org/TR/css-break-3/>

[CSS-BREAK-4]

Rossen Atanassov; Elika Etemad. *CSS Fragmentation Module Level 4*. 18 December 2018. WD. URL: <https://www.w3.org/TR/css-break-4/>

[CSS-FLEXBOX-1]

Tab Atkins Jr.; et al. *CSS Flexible Box Layout Module Level 1*. 19 November 2018. CR. URL: <https://www.w3.org/TR/css-flexbox-1/>

[CSS-FONTS-4]

John Daggett; Myles Maxfield; Chris Lilley. *CSS Fonts Module Level 4*. 21 December 2021. WD. URL: <https://www.w3.org/TR/css-fonts-4/>

[CSS-INLINE-3]

Dave Cramer; Elika Etemad; Steve Zilles. *CSS Inline Layout Module Level 3*. 27 August 2020. WD. URL: <https://www.w3.org/TR/css-inline-3/>

[CSS-LISTS-3]

Elika Etemad; Tab Atkins Jr.. *CSS Lists and Counters Module Level 3*. 17 November 2020. WD. URL: <https://www.w3.org/TR/css-lists-3/>

[CSS-PAGE-3]

Elika Etemad; Simon Sapin. *CSS Paged Media Module Level 3*. 18 October 2018. WD. URL: <https://www.w3.org/TR/css-page-3/>

[CSS-SIZING-3]

Tab Atkins Jr.; Elika Etemad. *CSS Box Sizing Module Level 3*. 17 December 2021. WD. URL: <https://www.w3.org/TR/css-sizing-3/>

[CSS-TEXT-3]

Elika Etemad; Koji Ishii; Florian Rivoal. *CSS Text Module Level 3*. 22 April 2021. CR. URL: <https://www.w3.org/TR/css-text-3/>

[CSS-WRITING-MODES-4]

Elika Etemad; Koji Ishii. *CSS Writing Modes Level 4*. 30 July 2019. CR. URL: <https://www.w3.org/TR/css-writing-modes-4/>

[SVG11]

Erik Dahlström; et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. 16 August 2011. REC. URL:

<https://www.w3.org/TR/SVG11/>

§ Property Index

Name	Value	Initial	Applies to	Inh.	%ages	Animation type	Canonical order	Computed value
<u>all</u>	initial inherit unset revert	see individual properties	see individual properties	see individual properties	see individual properties	see individual properties	per grammar	see individual properties