CSS Cascading and Inheritance Level 3

W3C Recommendation, 11 February 2021

This version:
https://www.w3.org/TR/2021/REC-css-cascade-3-20210211/

Latest published version:
https://www.w3.org/TR/css-cascade-3/

Editor's Draft:
https://drafts.csswg.org/css-cascade-3/

Previous Versions:
https://www.w3.org/TR/2020/PR-css-cascade-3-20201222/
https://www.w3.org/TR/2020/CR-css-cascade-3-20200817/
https://www.w3.org/TR/2018/CR-css-cascade-3-20180828/
https://www.w3.org/TR/2016/CR-css-cascade-3-20160519/
https://www.w3.org/TR/2015/CR-css-cascade-3-20150416/
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/

Implementation Report:
https://drafts.csswg.org/css-cascade-3/implementation-report.html

Test Suite:
http://test.csswg.org/suites/css-cascade-3_dev/nightly-unstable/

Issue Tracking:
CSSWG Issues Repository
Disposition of Comments

Editors:
Elika J. Etemad / fantasai (Invited Expert)
Tab Atkins Jr. (Google)

Suggest an Edit for this Spec:
GitHub Editor

Please check the **errata** for any errors or issues reported since publication.

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.

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 **Recommendation**.

A W3C Recommendation is a specification that, after extensive consensus-building, has received the endorsement of the W3C and its Members. W3C recommends the wide deployment of this specification as a standard for the Web.

Please send feedback by filing issues in GitHub (preferred), including the spec code "[css-cascade]" in the title, like this: "[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 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
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 Resetting All Properties: the 'all' property

4 Value Processing
4.1 Declared Values
4.2 Cascaded Values
4.3 Specified Values
4.4 Computed Values
4.5 Used Values
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

8 Changes
8.1 Changes Since the 28 August 2018 Candidate Recommendation
8.2 Changes Since the 19 May 2016 Candidate Recommendation
8.3 Changes Since the 3 October 2013 Candidate Recommendation
8.4 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

Index
Terms defined by this specification
Terms defined by reference

References
Normative References
Informative References

Property Index

§ 1. Introduction

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.

§ 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> ] <media-query-list>? ;
```

Where the <url> or <string> gives the URL of the style sheet to be imported, and the optional <media-query-list> (the ***import conditions***) states the conditions under which it applies.

If a <string> is provided, it must be interpreted as a <url> with the same value.

> EXAMPLE 2
> 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-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 an '@media' block with the given media query.

> EXAMPLE 3
> The following rules illustrate how '@import' rules can be made media-dependent:
>
> ```
> @import url("fineprint.css") print;
> @import url("bluish.css") projection, tv;
> @import url("narrow.css") handheld and (max-width: 400px);
> ```

User agents may therefore avoid fetching a media-dependent import as long as the media query does not match.

The evaluation and full syntax of the import conditions is defined by the Media Queries specification [MEDIAQ].

§ 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 origin of an imported style sheet is the 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]

§ 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, or the host document is in quirks mode and has the same origin as the imported style sheet, the type of the linked resource is `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 4

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. Resetting All Properties: the 'all' property

Name:	***'all'***
Value:	initial | inherit | unset
Initial:	see individual properties
Applies to:	see individual properties
Inherited:	see individual properties
Percentages:	see individual properties
Computed value:	see individual properties

Animation type:	see individual properties
Canonical order:	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 5

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 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.

§ 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”.

§ 4.2. 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 6

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 7
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 8
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—as noted in its "Applies to" line—then it has no direct formatting effect on that type of box or element, and therefore has no used value for that property.

EXAMPLE 9
For example, the 'flex' property has no used value on elements that aren't flex items.

Note: Some properties that do not apply to certain elements or boxes may still have *indirect* formatting effects: their computed value may be taken into account in the computation of other properties or units that do have an effect on the element or box; inherited properties are also often set on elements that they don't apply to in order to get them to inherit into descendants to which they do apply, including cases where these descendants are anonymous boxes.

EXAMPLE 10
Even though 'writing-mode' and 'text-orientation' do not apply to table rows, 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 11
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.

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)	**'text-align'**	`text-align: left`	'left'	'left'	'left'	'left'	'left'
(b)	**'border-top-width'**, **'border-right-width'**, **'border-bottom-width'**, **'border-left-width'**	`border-width: inherit`	'inherit'	'4.2px'	'4.2px'	'4.2px'	'4px'
(c)	**'width'**	(none)	(none)	'auto' (initial value)	'auto'	'120px'	'120px'
(d)	**'list-style-position'**	`list-style-position: inherit`	'inherit'	'inside'	'inside'	'inside'	'inside'
(e)	**'list-style-position'**	`list-style-position: initial`	'initial'	'outside' (initial value)	'outside'	'outside'	'outside'
(f)	**'font-size'**	`font-size: 1.2em`	'1.2em'	'1.2em'	'14.1px'	'14.1px'	'14px'
(g)	**'width'**	`width: 80%`	'80%'	'80%'	'80%'	'354.2px'	'354px'
(h)	**'width'**	`width: auto`	'auto'	'auto'	'auto'	'134px'	'134px'
(i)	**'height'**	`height: auto`	'auto'	'auto'	'auto'	'176px'	'176px'
(j)	**'page-break-after'**	(none)	(none)	'auto' (initial value)	'auto'	'auto'	'auto'
(k)	**'orphans'**	`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]
- 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.

¶ **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 12
>
> ```
> [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 13

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 either enter the cascade as UA-origin rules or be treated as author-origin rules with a specificity of zero placed at the start of the author style sheet. 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 level.

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-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.

Pseudo-elements inherit according to the fictional tag sequence described for each pseudo-element. [SELECT]

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 Level 3 [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).

8. Changes

8.1. Changes Since the 22 December 2020 Proposed Recommendation

The following changes were made to this specification since the 22 December 2020 Proposed Recommendation:

- Correction of a typo
- Status and styling changes for W3C Recommendation

§ 8.2. Changes Since the 28 August 2018 Candidate Recommendation

The following changes were made to this specification since the 28 August 2018 Candidate Recommendation:

- Clarify that '@import' ordering requirements are only in consideration of valid rules.
- Clarify that computed values are not serialization strings, but abstract data.

§ 8.3. Changes Since the 19 May 2016 Candidate Recommendation

The following non-trivial changes were made to this specification since the 19 May 2016 Candidate Recommendation:

¶ - 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].

¶ - Called out two exceptions in which importing a style sheet is different from merely inserting its contents.

> 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.

¶ - Removed any mention of scoped stylesheets, as the feature was removed from HTML. (Issue 637)

¶ - Removed any mention of the obsolete "override" origin, originally defined by DOM Level 2 Style and later abandoned.

A Disposition of Comments is available.

§ 8.4. Changes Since the 3 October 2013 Candidate Recommendation

The following changes were made to this specification since the 3 October 2013 Candidate Recommendation:

- Defined environment encoding of imported style sheets.

> The environment encoding of an imported style sheet is the encoding of the style sheet that imported it. [css-syntax-3]

- Referenced [css-syntax-3] for syntax of '!important' rules.

> A declaration is important if it has a '!important' annotation, as defined by [css-syntax-3] .

- Explained reset-only sub-properties and clarified that they also get affected by a CSS-wide keyword value in the shorthand declaration.

> 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.)

A Disposition of Comments is available.

§ 8.5. 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, Simon Sapin, 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 14
>
> 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

actual value, in §4.6
all, in §3.1
Animation Origin, in §6.2
applies to, in §4.5
apply to, in §4.5
author origin, in §6.2
author-origin, in §6.2
author style sheet, in §6.2
cascade, in §6
cascaded value, in §4.2
cascade origin, in §6.2
computed value, in §4.4
declared value, in §4.1
@import, in §2
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
longhand, in §3
longhand property, in §3
normal, in §6.3
origin, in §6.2
output of the cascade, in §6.1
reset-only sub-property, in §3
shorthand, in §3
shorthand property, in §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 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-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-top-width

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

- break-before
- orphans

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

- page

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

- display type
- inline box

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

- flex
- flex item

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

- font
- font-family
- font-size
- font-size-adjust
- font-style
- font-variant
- font-weight

[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-sizing-3] defines the following terms:

- height
- width

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

- @charset
- environment encoding

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

- text-align
- text-indent
- text-transform

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

- <length>
- <string>
- <url>
- ?
- ch
- css-wide keywords
- em
- ex
- 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:

- bolder
- display
- italic
- line-height
- page-break-after
- red
- sans-serif

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

- @media

[CSSOM] defines the following terms:

- getComputedStyle(elt)

[FETCH] defines the following terms:

- cors protocol

[HTML] defines the following terms:

- p
- s

[MEDIAQ] defines the following terms:

- <media-query-list>
- <media-query>

[selectors-4] defines the following terms:

- pseudo-element

§ References

§ Normative References

[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-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/

[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.; Elika Etemad. CSS Values and Units Module Level 3. 6 June 2019. CR. URL: https://www.w3.org/TR/css-values-3/

[CSS-VARIABLES-1]
Tab Atkins Jr.. CSS Custom Properties for Cascading Variables Module Level 1. 3 December 2015. CR. URL: https://www.w3.org/TR/css-variables-1/

[CSS-WRITING-MODES-3]
Elika 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/

[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/

[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. 21 July 2020. 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://tools.ietf.org/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/

§ Informative References

[CSS-ALIGN-3]
Elika Etemad; Tab Atkins Jr.. CSS Box Alignment Module Level 3. 21 April 2020. WD. URL: https://www.w3.org/TR/css-align-3/

[CSS-BACKGROUNDS-3]
Bert Bos; Elika Etemad; Brad Kemper. CSS Backgrounds and Borders Module Level 3. 17 October 2017. CR. URL: https://www.w3.org/TR/css-backgrounds-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-DISPLAY-3]
Tab Atkins Jr.; Elika Etemad. CSS Display Module Level 3. 18 December 2020. CR. URL: https://www.w3.org/TR/css-display-3/

[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-3]
John Daggett; Myles Maxfield; Chris Lilley. CSS Fonts Module Level 3. 20 September 2018. REC. URL: https://www.w3.org/TR/css-fonts-3/

[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-PSEUDO-4]
Daniel Glazman; Elika Etemad; Alan Stearns. CSS Pseudo-Elements Module Level 4. 25 February 2019. WD. URL: https://www.w3.org/TR/css-pseudo-4/

[CSS-SIZING-3]
Tab Atkins Jr.; Elika Etemad. CSS Box Sizing Module Level 3. 18 December 2020. WD. URL: https://www.w3.org/TR/css-sizing-3/

[CSS-TEXT-3]
Elika Etemad; Koji Ishii; Florian Rivoal. CSS Text Module Level 3. 19 November 2020. WD. 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/

[CSSOM]
Simon Pieters; Glenn Adams. CSS Object Model (CSSOM). 17 March 2016. WD. URL: https://www.w3.org/TR/cssom-1/

[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
'all'	initial | inherit | unset	see individual properties	see individual properties	see individual properties	see individual properties	see individual properties	per grammar	see individual properties

↥