CSS Values and Units Module Level 3

W3C Candidate Recommendation Snapshot, 01 December 2022

▼ **More details about this document**

This version:

https://www.w3.org/TR/2022/CR-css-values-3-20221201/

Latest published version:

https://www.w3.org/TR/css-values-3/

Editor's Draft:

https://drafts.csswg.org/css-values-3/

History:

https://www.w3.org/standards/history/css-values-3

Implementation Report:

https://drafts.csswg.org/css-values-3/implementation-report.html

Test Suites:

http://test.csswg.org/suites/css-values-3_dev/nightly-unstable/

https://wpt.fyi/results/css/

Feedback:

CSSWG Issues Repository

Editors:

Tab Atkins (Google)

fantasai

Former Editor:

Håkon Wium Lie (Opera Software)

Suggest an Edit for this Spec:

GitHub Editor

Abstract

This CSS module describes the common values and units that CSS properties accept and the syntax

used for describing them in CSS property definitions.

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/TR/.

This document was published by the CSS Working Group as a **Candidate Recommendation Snapshot** using the Recommendation track. Publication as a Candidate Recommendation does not imply endorsement by W3C and its 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 implementations. This document is intended to become a W3C Recommendation; it will remain a Candidate Recommendation at least until 31 January 2023 to gather additional feedback.

Please send feedback by filing issues in GitHub (preferred), including the spec code "css-values" in the title, like this: "[css-values] *...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 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

The value definition field of each CSS property can contain keywords, data types (which appear between ‘<’ and ‘>’), and information on how they can be combined. Generic data types (<length> being the most widely used) that can be used by many properties are described in this specification, while more specific data types (e.g., <spacing-limit>) are described in the corresponding modules.

§ 1.1. Module Interactions

This module replaces and extends the data type definitions in [CSS21] sections 1.4.2.1, 4.3, and A.2.

§ 2. Value Definition Syntax

The ***value definition syntax*** described here is used to define the set of valid values for CSS properties (and the valid syntax of many other parts of CSS). A value so described can have one or more components.

§ 2.1. Component Value Types

Component value types are designated in several ways:

1. keyword values (such as 'auto', 'disc', etc.), which appear literally, without quotes (e.g. `auto`)
2. basic data types, which appear between '<' and '>' (e.g., <length>, <percentage>, etc.). For numeric data types, this type notation can annotate any range restrictions using the bracketed range notation described below.
3. types that have the same range of values as a property bearing the same name (e.g., <'border-width'>, <'background-attachment'>, etc.). In this case, the type name is the property name (complete with quotes) between the brackets. Such a type does *not* include CSS-wide keywords such as 'inherit', and also does not include any top-level comma-separated-list multiplier (i.e. if a property named 'pairing' is defined as '[<custom-ident> <integer>?]#', then '<'pairing'>' is equivalent to '[<custom-ident> <integer>?]', not '[<custom-ident> <integer>?]#').
4. non-terminals that do not share the same name as a property. In this case, the non-terminal name appears between '<' and '>', as in <spacing-limit>. Notice the distinction between <border-width> and <'border-width'>: the latter is defined as the value of the 'border-width' property, the former requires an explicit expansion elsewhere. The definition of a non-terminal is typically located near its first appearance in the specification.

Some property value definitions also include the slash (/), the comma (,), and/or parentheses as literals. These represent their corresponding tokens. Other non-keyword literal characters that may appear in a component value, such as "+", must be written enclosed in single quotes.

Commas **specified in the grammar are implicitly omissible** in some circumstances, when used to separate optional terms in the grammar. Within a top-level list in a property or other CSS value, or a function's argument list, a comma specified in the grammar must be omitted if:

- all items preceding the comma have been omitted
- all items following the comma have been omitted
- multiple commas would be adjacent (ignoring white space/comments), due to the items between the commas being omitted.

EXAMPLE 1

For example, if a function can accept three arguments in order, but all of them are optional, the grammar can be written like:

```
example( first? , second? , third? )
```

Given this grammar, writing 'example(first, second, third)' is valid, as is 'example(first, second)' or 'example(first, third)' or 'example(second)'. However, 'example(first, , third)' is invalid, as one of those commas are no longer separating two options; similarly, 'example(,second)' and 'example(first,)' are invalid. 'example(first second)' is also invalid, as commas are still required to actually separate the options.

If commas were not implicitly omittable, the grammar would have to be much more complicated to properly express the ways that the arguments can be omitted, greatly obscuring the simplicity of the feature.

All CSS properties also accept the CSS-wide keyword values as the sole component of their property value. For readability these are not listed explicitly in the property value syntax definitions. For example, the full value definition of 'border-color' under CSS Cascading and Inheritance Level 3 is `<color>{1,4} | inherit | initial | unset` (even though it is listed as `<color>{1,4}`).

Note: This implies that, in general, combining these keywords with other component values in the same declaration results in an invalid declaration. For example, 'background: url(corner.png) no-repeat, inherit;' is invalid.

§ 2.2. Component Value Combinators

Component values can be arranged into property values as follows:

- Juxtaposing components means that all of them must occur, in the given order.
- A double ampersand (**&&**) separates two or more components, all of which must occur, in any order.
- A double bar (||) separates two or more options: one or more of them must occur, in any order.

- A bar (|) separates two or more alternatives: exactly one of them must occur.
- Brackets ([ ]) are for grouping.

Juxtaposition is stronger than the double ampersand, the double ampersand is stronger than the double bar, and the double bar is stronger than the bar. Thus, the following lines are equivalent:

```
  a b   |   c ||   d &&   e f
[ a b ] | [ c || [ d && [ e f ]]]
```

For reorderable combinators (||, &&), ordering of the grammar does not matter: components in the same grouping may be interleaved in any order. Thus, the following lines are equivalent:

```
a || b || c
b || a || c
```

§ 2.3. Component Value Multipliers

Every type, keyword, or bracketed group may be followed by one of the following modifiers:

- An asterisk (*) indicates that the preceding type, word, or group occurs zero or more times.
- A plus (+) indicates that the preceding type, word, or group occurs one or more times.
- A question mark (*?*) indicates that the preceding type, word, or group is optional (occurs zero or one times).
- A single number in curly braces (*{***A***}*) indicates that the preceding type, word, or group occurs *A* times.
- A comma-separated pair of numbers in curly braces (*{***A**,**B***}*) indicates that the preceding type, word, or group occurs at least *A* and at most *B* times. The *B* may be omitted ({*A*,}) to indicate that there must be at least *A* repetitions, with no upper bound on the number of repetitions.
- A hash mark (#) indicates that the preceding type, word, or group occurs one or more times, separated by comma tokens (which may optionally be surrounded by white space and/or comments). It may optionally be followed by the curly brace forms, above, to indicate precisely how many times the repetition occurs, like '<length>#{1,4}'.
- An exclamation point (*!*) after a group indicates that the group is required and must produce at least one value; even if the grammar of the items within the group would otherwise allow the entire contents to be omitted, at least one component value must not be omitted.

For repeated component values (indicated by '*', '+', or '#'), UAs must support at least 20 repetitions of the component. If a property value contains more than the supported number of

repetitions, the declaration must be ignored as if it were invalid.

§ 2.4. Combinator and Multiplier Patterns

There are a small set of common ways to combine multiple independent component values in particular numbers and orders. In particular, it's common to want to express that, from a set of component value, the author must select zero or more, one or more, or all of them, and in either the order specified in the grammar or in any order.

All of these can be easily expressed using simple patterns of combinators and multipliers:

	in order	**any order**
zero or more	`A? B? C?`	`A? || B? || C?`
one or more	`[ A? B? C? ]!`	`A || B || C`
all	`A B C`	`A && B && C`

Note that all of the "any order" possibilities are expressed using combinators, while the "in order" possibilities are all variants on juxtaposition.

§ 2.5. Component Values and White Space

Unless otherwise specified, white space and/or comments may appear before, after, and/or between components combined using the above combinators and multipliers.

> Note: In many cases, spaces will in fact be *required* between components in order to distinguish them from each other. For example, the value ‘1em2em’ would be parsed as a single <dimension-token> with the number ‘1’ and the identifier ‘em2em’, which is an invalid unit. In this case, a space would be required before the ‘2’ to get this parsed as the two lengths ‘1em’ and ‘2em’.

§ 2.6. Property Value Examples

Below are some examples of properties with their corresponding value definition fields

EXAMPLE 2

Property	Value definition field	Example value
'orphans'	<integer>	'3'
'text-align'	left | right | center | justify	'center'
'padding-top'	<length> | <percentage>	'5%'
'outline-color'	<color> | invert	'#fefefe'
'text-decoration'	none | underline || overline || line-through || blink	'overline underline'
'font-family'	[<family-name> | <generic-family>]#	'"Gill Sans", Futura, sans-serif'
'border-width'	[<length> | thick | medium | thin]{1,4}	'2px medium 4px'
'box-shadow'	[inset? && <length>{2,4} && <color>?]# | none	'3px 3px rgba(50%, 50%, 50%, 50%), lemonchiffon 0 0 4px inset'

§ 3. Textual Data Types

The ***textual data types*** include various keywords and identifiers as well as strings (<string>) and URLs (<url>).

CSS ***identifiers***, generically denoted by ***'<ident>'***, consist of a sequence of characters conforming to the <ident-token> grammar. [CSS-SYNTAX-3] Identifiers cannot be quoted; otherwise they would be interpreted as strings. CSS properties accept two classes of identifiers: pre-defined keywords and author-defined identifiers.

Note: The <ident> production is not meant for property value definitions—<custom-ident> should be used instead. It is provided as a convenience for defining other syntactic constructs.

§ 3.1. Pre-defined Keywords

In the value definition fields, ***keywords*** with a pre-defined meaning appear literally. Keywords are CSS identifiers and are interpreted ASCII case-insensitively (i.e., [a-z] and [A-Z] are equivalent).

EXAMPLE 3

For example, here is the value definition for the 'border-collapse' property:

```
Value: collapse | separate
```

And here is an example of its use:

```
table { border-collapse: separate }
```

3.1.1. CSS-wide keywords: 'initial', 'inherit' and 'unset'

As defined above, all properties accept the ***CSS-wide keywords***, which represent value computations common to all CSS properties. These keywords are normatively defined in the CSS Cascading and Inheritance Module.

Other CSS specifications can define additional CSS-wide keywords.

3.2. Author-defined Identifiers: the <custom-ident> type

Some properties accept arbitrary author-defined identifiers as a component value. This generic data type is denoted by ***'<custom-ident>'***, and represents any valid CSS identifier that would not be misinterpreted as a pre-defined keyword in that property's value definition. Such identifiers are fully case-sensitive, even in the ASCII range (e.g. 'example' and 'EXAMPLE' are two different, unrelated user-defined identifiers).

The CSS-wide keywords are not valid <custom-ident>s. The 'default' keyword is reserved and is also not a valid <custom-ident>. Specifications using <custom-ident> must specify clearly what other keywords are excluded from <custom-ident>, if any—for example by saying that any pre-defined keywords in that property's value definition are excluded. Excluded keywords are excluded in all ASCII case permutations.

When parsing positionally-ambiguous keywords in a property value, a <custom-ident> production can only claim the keyword if no other unfulfilled production can claim it.

EXAMPLE 4

For example, the shorthand declaration ‘animation: ease-in ease-out’ is equivalent to the longhand declarations ‘animation-timing-function: ease-in; animation-name: ease-out;’. ‘ease-in’ is claimed by the <easing-function> production belonging to ‘animation-timing-function’, leaving ‘ease-out’ to be claimed by the <custom-ident> production belonging to ‘animation-name’.

Note: When designing grammars with <custom-ident>, the <custom-ident> should always be "positionally unambiguous", so that it’s impossible to conflict with any keyword values in the property.

§ 3.3. Quoted Strings: the <string> type

Strings are denoted by ***‘<string>’*** and consist of a sequence of characters delimited by double quotes or single quotes. They correspond to the <string-token> production in the CSS Syntax Module [CSS-SYNTAX-3].

EXAMPLE 5

Double quotes cannot occur inside double quotes, unless escaped (as `"\""` or as `"\22"`). Analogously for single quotes (`'\''` or `'\27'`).

```
content: "this is a 'string'.";
content: "this is a \"string\".";
content: 'this is a "string".';
content: 'this is a \'string\'.'
```

It is possible to break strings over several lines, for aesthetic or other reasons, but in such a case the newline itself has to be escaped with a backslash (\). The newline is subsequently removed from the string. For instance, the following two selectors are exactly the same:

EXAMPLE 6

```
a[title="a not s\
o very long title"] {/*...*/}
a[title="a not so very long title"] {/*...*/}
```

Since a string cannot directly represent a newline, to include a newline in a string, use the escape "\A". (Hexadecimal A is the line feed character in Unicode (U+000A), but represents the generic notion of "newline" in CSS.)

§ 3.4. Resource Locators: the <url> type

The ***'url()'*** functional notation, denoted by <url>, represents a ***URL***, which is a pointer to a resource. The typical syntax of a <url> is:

```
<url> = url( <string> <url-modifier>* )
```

EXAMPLE 7
Below is an example of a URL being used as a background image:

```
body { background: url("http://www.example.com/pinkish.gif") }
```

A <url> may alternately be written without quotation marks around the URL itself, in which case it is specially-parsed as a <url-token> [CSS-SYNTAX-3].

EXAMPLE 8
For example, the following declarations act identically:

```
background: url("http://www.example.com/pinkish.gif");
background: url(http://www.example.com/pinkish.gif);
```

Note: This unquoted syntax cannot accept a <url-modifier> argument and has extra escaping requirements: parentheses, whitespace characters, single quotes (') and double quotes (") appearing in a URL must be escaped with a backslash, e.g. 'url(open\(parens)', 'url(close\)parens)'. (In quoted <string> 'url()'s, only newlines and the character used to quote the string need to be escaped.) Depending on the type of URL, it might also be possible to write these characters as URL-escapes (e.g. 'url(open%28parens)' or 'url(close%29parens)') as described in [URL].

Some CSS contexts (such as '@import') also allow a <url> to be represented by a bare <string>, without the 'url()' wrapper. In such cases the string behaves identically to a 'url()' function containing that string.

EXAMPLE 9
For example, the following statements are identical:

```
@import url("base-theme.css");
@import "base-theme.css";
```

3.4.1. Relative URLs

In order to create modular style sheets that are not dependent on the absolute location of a resource, authors should use relative URLs. Relative URLs (as defined in [URL]) are resolved to full URLs using a base URL. RFC 3986, section 3, defines the normative algorithm for this process. For CSS style sheets, the base URL is that of the style sheet itself, not that of the styled source document. Style sheets embedded within a document have the base URL associated with their container.

> Note: For HTML documents, the base URL is mutable.

When a <url> appears in the computed value of a property, it is resolved to an absolute URL, as described in the preceding paragraph. The computed value of a URL that the UA cannot resolve to an absolute URL is the specified value.

> EXAMPLE 10
> For example, suppose the following rule:
>
> ```
> body { background: url("tile.png") }
> ```
>
> is located in a style sheet designated by the URL:
>
> ```
> http://www.example.org/style/basic.css
> ```
>
> The background of the source document's `<body>` will be tiled with whatever image is described by the resource designated by the URL:
>
> ```
> http://www.example.org/style/tile.png
> ```
>
> The same image will be used regardless of the URL of the source document containing the `<body>`.

3.4.1.1. Fragment URLs

To work around some common eccentricities in browser URL handling, CSS has special behavior for fragment-only urls.

If a 'url()''s value starts with a U+0023 NUMBER SIGN (#) character, parse it as per normal for URLs, but additionally set the ***local url flag*** of the 'url()'.

When matching a 'url()' with the local url flag set, ignore everything but the URL's fragment, and

resolve that fragment against the current document that relative URLs are resolved against. This reference must always be treated as same-document (rather than cross-document).

When serializing a 'url()' with the local url flag set, it must serialize as just the fragment.

▸ **What "browser eccentricities"?**

§ 3.4.2. Empty URLs

If the value of the 'url()' is the empty string (like 'url("")' or 'url()'), the url must resolve to an invalid resource (similar to what the url 'about:invalid' does).

Its computed value is 'url("")', and it must serialize as such.

Note: This matches the behavior of empty urls for embedded resources elsewhere in the web platform, and avoids excess traffic re-requesting the stylesheet or host document due to editing mistakes leaving the 'url()' value empty, which are almost certain to be invalid resources for whatever the 'url()' shows up in. Linking on the web platform *does* allow empty urls, so if/when CSS gains some functionality to control hyperlinks, this restriction can be relaxed in those contexts.

§ 3.4.3. URL Modifiers

The 'url()' function supports specifying additional ***'<url-modifier>'***s, which change the meaning or the interpretation of the URL somehow. A <url-modifier> is either an <ident> or a functional notation.

This specification does not define any <url-modifier>s, but other specs may do so.

Note: A <url> that is either unquoted or not wrapped in 'url()' notation cannot accept any <url-modifier>s.

§ 4. Numeric Data Types

Numeric data types are used to represent quantities, indexes, positions, and other such values. Although many syntactic variations can exist in expressing the quantity (numeric aspect) in a given numeric value, the specified and computed value do not distinguish these variations: they represent the value's abstract quantity, not its syntactic representation.

The ***numeric data types*** include <integer>, <number>, <percentage>, and various dimensions

including <length>, <angle>, <time>, <frequency>, and <resolution>.

> Note: While general-purpose dimensions are defined here, some other modules define additional data types (e.g. [css-grid-1] introduces 'fr' units) whose usage is more localized.

§ 4.1. Range Restrictions and Range Definition Notation

Properties can restrict numeric values to some range. If the value is outside the allowed range, then unless otherwise specified, the declaration is invalid and must be ignored. Range restrictions can be annotated in the numeric type notation using ***CSS bracketed range notation***—`[min,max]`—within the angle brackets, after the identifying keyword, indicating a closed range between (and including) *min* and *max*. For example, <integer [0,10]> indicates an integer between '0' and '10', inclusive, while <angle [0,180deg]> indicates an angle between '0deg' and '180deg' (expressed in any unit).

> Note: CSS values generally do not allow open ranges; thus only square-bracket notation is used.

CSS theoretically supports infinite precision and infinite ranges for all value types; however in reality implementations have finite capacity. UAs should support reasonably useful ranges and precisions. Range extremes that are ideally unlimited are indicated using ∞ or −∞ as appropriate. For example, <length [0,∞]> indicates a non-negative length.

If no range is indicated, either by using the bracketed range notation or in the property description, then `[-∞,∞]` is assumed.

Values of −∞ or ∞ must be written without units, even if the value type uses units. Values of '0' *can* be written without units, even if the value type doesn't allow "unitless zeroes" (such as <time>).

> Note: At the time of writing, the bracketed range notation is new; thus in most CSS specifications any range limitations are described only in prose. (For example, "Negative values are not allowed" or "Negative values are invalid" indicate a `[0,∞]` range.) This does not make them any less binding.

§ 4.2. Integers: the <integer> type

Integer values are denoted by ***'<integer>'***.

When written literally, an ***integer*** is one or more decimal digits '0' through '9' and corresponds to

a subset of the <number-token> production in the CSS Syntax Module [CSS-SYNTAX-3]. The first digit of an integer may be immediately preceded by '-' or '+' to indicate the integer's sign.

§ 4.3. Real Numbers: the <number> type

Number values are denoted by ***'<number>'***, and represent real numbers, possibly with a fractional component.

When written literally, a ***number*** is either an integer, optionally, it can be concluded by the letter "e" or "E" followed by an integer indicating the base-ten exponent in scientific notation. It corresponds to the <number-token> production in the CSS Syntax Module [CSS-SYNTAX-3]. As with integers, the first character of a number may be immediately preceded by '-' or '+' to indicate the number's sign.

§ 4.4. Numbers with Units: dimension values

The general term ***dimension*** refers to a number with a unit attached to it; and is denoted by ***'<dimension>'***.

When written literally, a dimension is a number immediately followed by a unit identifier, which is an identifier. It corresponds to the <dimension-token> production in the CSS Syntax Module [CSS-SYNTAX-3]. Like keywords, unit identifiers are ASCII case-insensitive.

CSS uses <dimension>s to specify distances (<length>), durations (<time>), frequencies (<frequency>), resolutions (<resolution>), and other quantities.

§ 4.4.1. Compatible Units

When serializing computed values [CSSOM], ***compatible units*** (those related by a static multiplicative factor, like the 96:1 factor between 'px' and 'in', or the computed 'font-size' factor between 'em' and 'px') are converted into a single ***canonical unit***. Each group of compatible units defines which among them is the canonical unit that will be used for serialization.

When serializing resolved values that are used values, all value types (percentages, numbers, keywords, etc.) that represent lengths are considered compatible with lengths. Likewise any future API that returns used values must consider any values that represent distances/durations /frequencies/etc. as compatible with the relevant class of dimensions, and canonicalize accordingly.

§ 4.5. Percentages: the <percentage> type

Percentage values are denoted by ***'<percentage>'***, and indicates a value that is some fraction of another reference value.

When written literally, a ***percentage*** consists of a number immediately followed by a percent sign '%'. It corresponds to the <percentage-token> production in the CSS Syntax Module [CSS-SYNTAX-3].

Percentage values are always relative to another quantity, for example a length. Each property that allows percentages also defines the quantity to which the percentage refers. This quantity can be a value of another property for the same element, the value of a property for an ancestor element, a measurement of the formatting context (e.g., the width of a containing block), or something else.

§ 4.6. Mixing Percentages and Dimensions

In cases where a <percentage> can represent the same quantity as a dimension in the same component value position, and can therefore be combined with them in a 'calc()' expression, the following convenience notations may be used in the property grammar:

'<length-percentage>'
: Equivalent to `[ <length> | <percentage> ]`, where the <percentage> will resolve to a <length>.

'<frequency-percentage>'
: Equivalent to `[ <frequency> | <percentage> ]`, where the <percentage> will resolve to a <frequency>.

'<angle-percentage>'
: Equivalent to `[ <angle> | <percentage> ]`, where the <percentage> will resolve to an <angle>.

'<time-percentage>'
: Equivalent to `[ <time> | <percentage> ]`, where the <percentage> will resolve to a <time>.

EXAMPLE 11

For example, the 'width' property can accept a <length> or a <percentage>, both representing a measure of distance. This means that 'width: calc(500px + 50%);' is allowed—both values are converted to absolute lengths and added. If the containing block is '1000px' wide, then 'width: 50%;' is equivalent to 'width: 500px', and 'width: calc(50% + 500px)' thus ends up equivalent to 'width: calc(500px + 500px)' or 'width: 1000px'.

On the other hand, the second and third arguments of the 'hsl()' function can only be expressed as <percentage>s. Although 'calc()' productions are allowed in their place, they can only combine percentages with themselves, as in 'calc(10% + 20%)'.

Note: Specifications should never alternate <percentage> in place of a dimension in a grammar unless they are compatible.

Note: More <*type*-percentage> productions can be added in the future as needed. A <number-percentage> will never be added, as <number> and <percentage> can't be combined in 'calc()'.

§ 5. Distance Units: the <length> type

Lengths refer to distance measurements and are denoted by ***'<length>'*** in the property definitions. A length is a dimension.

For zero lengths the unit identifier is optional (i.e. can be syntactically represented as the <number> '0'). However, if a '0' could be parsed as either a <number> or a <length> in a property (such as 'line-height'), it must parse as a <number>.

Properties may restrict the length value to some range. If the value is outside the allowed range, the declaration is invalid and must be ignored.

While some properties allow negative length values, this may complicate the formatting and there may be implementation-specific limits. If a negative length value is allowed but cannot be supported, it must be converted to the nearest value that can be supported.

In cases where the used length cannot be supported, user agents must approximate it in the actual value.

There are two types of length units: relative and absolute.

§ 5.1. Relative Lengths

Relative length units specify a length relative to another length. Style sheets that use relative units can more easily scale from one output environment to another.

The relative units are:

Informative Summary of Relative Units

unit	relative to
'em'	font size of the element
'ex'	x-height of the element's font
'ch'	character advance of the "0" (ZERO, U+0030) glyph in the element's font
'rem'	font size of the root element
'vw'	1% of viewport's width
'vh'	1% of viewport's height
'vmin'	1% of viewport's smaller dimension
'vmax'	1% of viewport's larger dimension

Child elements do not inherit the relative values as specified for their parent; they inherit the computed values.

§ 5.1.1. Font-relative Lengths: the 'em', 'ex', 'ch', 'rem' units

The ***font-relative lengths*** refer to the font metrics of the element on which they are used—or, in the case of 'rem', the metrics of the root element.

'em unit'

Equal to the computed value of the 'font-size' property of the element on which it is used.

EXAMPLE 12

The rule:

```
h1 { line-height: 1.2em }
```

means that the line height of `h1` elements will be 20% greater than the font size of `h1` element. On the other hand:

```
h1 { font-size: 1.2em }
```

means that the font size of `h1` elements will be 20% greater than the computed font size inherited by `h1` elements.

'ex unit'

Equal to the used x-height of the first available font [CSS3-FONTS]. The x-height is so called because it is often equal to the height of the lowercase "x". However, an 'ex' is defined even for fonts that do not contain an "x". The x-height of a font can be found in different ways. Some fonts contain reliable metrics for the x-height. If reliable font metrics are not available, UAs may determine the x-height from the height of a lowercase glyph. One possible heuristic is to look at how far the glyph for the lowercase "o" extends below the baseline, and subtract that value from the top of its bounding box. In the cases where it is impossible or impractical to determine the x-height, a value of 0.5em must be assumed.

'ch unit'

Represents the typical advance measure of European alphanumeric characters, and measured as the used advance measure of the "0" (ZERO, U+0030) glyph in the font used to render it. (The ***advance measure*** of a glyph is its advance width or height, whichever is in the inline axis of the element.)

Note: This measurement is an approximation (and in monospace fonts, an exact measure) of a single narrow glyph's advance measure, thus allowing measurements based on an expected glyph count.

Note: The advance measure of a glyph depends on writing-mode and text-orientation as well as font settings, text-transform, and any other properties that affect glyph selection or orientation.

In the cases where it is impossible or impractical to determine the measure of the "0" glyph, it must be assumed to be 0.5em wide by 1em tall. Thus, the 'ch' unit falls back to '0.5em' in the general case, and to '1em' when it would be typeset upright (i.e. 'writing-mode' is 'vertical-

rl' or 'vertical-lr' and 'text-orientation' is 'upright').

'rem unit'

Equal to the computed value of 'font-size' on the root element.

When specified in the 'font-size' property of the root element, or in a document with no root element, '1rem' is equal to the initial value of the 'font-size' property.

When used outside the context of an element (such as in media queries), these units refer to the computed font metrics corresponding to the initial values of the 'font' property. When used in the value of the 'font-size' property on the element they refer to, these units refer to the computed font metrics of the parent element (or the computed font metrics corresponding to the initial values of the 'font' property, if the element has no parent).

The font-relative lengths are calculated in the absence of shaping.

§ 5.1.2. Viewport-percentage Lengths: the 'vw', 'vh', 'vmin', 'vmax' units

The ***viewport-percentage lengths*** are relative to the size of the initial containing block. When the height or width of the initial containing block is changed, they are scaled accordingly. However, any scrollbars are assumed not to exist.

For paged media, the exact definition of the viewport-percentage lengths is deferred to [CSS3PAGE].

'vw unit'

Equal to 1% of the width of the initial containing block.

> EXAMPLE 13
> In the example below, if the width of the viewport is 200mm, the font size of `h1` elements will be 16mm (i.e. (8×200mm)/100).
>
> ```
> h1 { font-size: 8vw }
> ```

'vh unit'

Equal to 1% of the height of the initial containing block.

'vmin unit'

Equal to the smaller of 'vw' or 'vh'.

'vmax unit'

Equal to the larger of 'vw' or 'vh'.

§ 5.2. Absolute Lengths: the 'cm', 'mm', 'Q', 'in', 'pt', 'pc', 'px' units

The ***absolute length units*** are fixed in relation to each other and anchored to some physical measurement. They are mainly useful when the output environment is known. The absolute units consist of the ***physical units*** ('in', 'cm', 'mm', 'pt', 'pc', 'Q') and the ***visual angle unit (pixel unit)*** ('px'):

unit	name	equivalence
'cm'	centimeters	1cm = 96px/2.54
'mm'	millimeters	1mm = 1/10th of 1cm
'Q'	quarter-millimeters	1Q = 1/40th of 1cm
'in'	inches	1in = 2.54cm = 96px
'pc'	picas	1pc = 1/6th of 1in
'pt'	points	1pt = 1/72nd of 1in
'px'	pixels	1px = 1/96th of 1in

EXAMPLE 14

```
h1 { margin: 0.5in }      /* inches  */
h2 { line-height: 3cm }   /* centimeters */
h3 { word-spacing: 4mm }  /* millimeters */
h3 { letter-spacing: 1Q } /* quarter-millimeters */
h4 { font-size: 12pt }    /* points */
h4 { font-size: 1pc }     /* picas */
p  { font-size: 12px }    /* px */
```

All of the absolute length units are compatible, and 'px' is their canonical unit.

For a CSS device, these dimensions are ***anchored*** either

i. by relating the physical units to their physical measurements, or

ii. by relating the pixel unit to the reference pixel.

For print media at typical viewing distances, the anchor unit should be one of the physical units (inches, centimeters, etc). For screen media (including high-resolution devices), low-resolution devices, and devices with unusual viewing distances, it is recommended instead that the anchor

unit be the pixel unit. For such devices it is recommended that the pixel unit refer to the whole number of device pixels that best approximates the reference pixel.

> Note: If the anchor unit is the pixel unit, the physical units might not match their physical measurements. Alternatively if the anchor unit is a physical unit, the pixel unit might not map to a whole number of device pixels.

> Note: This definition of the pixel unit and the physical units differs from the earlier editions of CSS1 and CSS2. In particular, in previous versions of CSS the pixel unit and the physical units were not related by a fixed ratio: the physical units were always tied to their physical measurements while the pixel unit would vary to most closely match the reference pixel. (This unfortunate change was made because too much existing content relies on the assumption of 96dpi, and breaking that assumption broke the content.)

> Note: Units are ASCII case-insensitive and serialize as lower case, for example 1Q serializes as 1q.

The ***reference pixel*** is the visual angle of one pixel on a device with a device pixel density of 96dpi and a distance from the reader of an arm's length. For a nominal arm's length of 28 inches, the visual angle is therefore about 0.0213 degrees. For reading at arm's length, 1px thus corresponds to about 0.26 mm (1/96 inch).

The image below illustrates the effect of viewing distance on the size of a reference pixel: a reading distance of 71 cm (28 inches) results in a reference pixel of 0.26 mm, while a reading distance of 3.5 m (12 feet) results in a reference pixel of 1.3 mm.

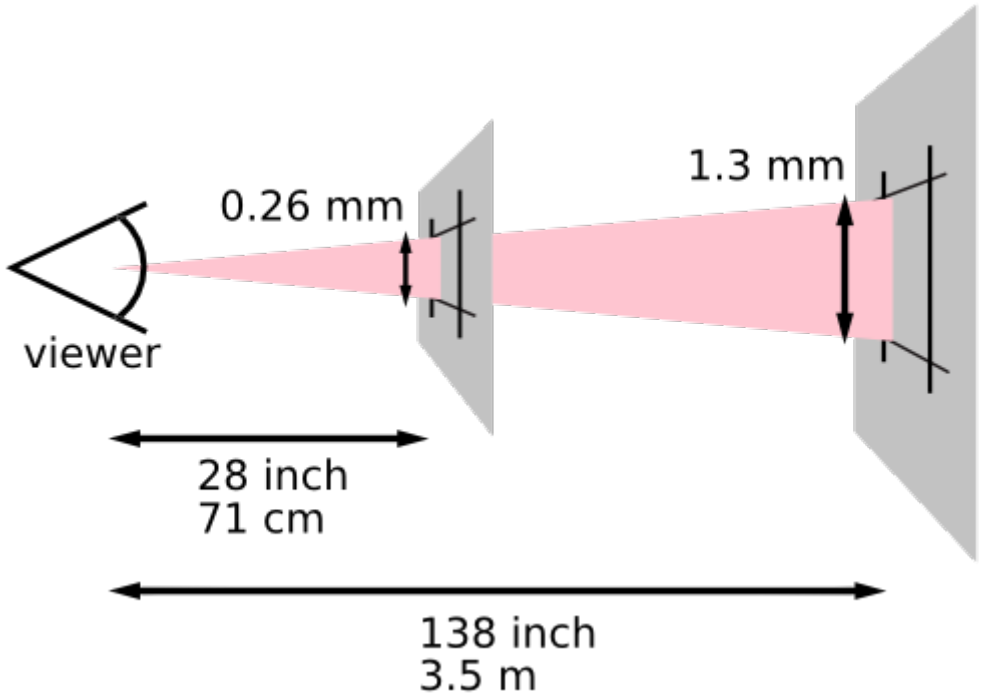

Figure 1 *Showing that pixels must become larger if the viewing distance increases*

This second image illustrates the effect of a device's resolution on the pixel unit: an area of 1px by 1px is covered by a single dot in a low-resolution device (e.g. a typical computer display), while the same area is covered by 16 dots in a higher resolution device (such as a printer).

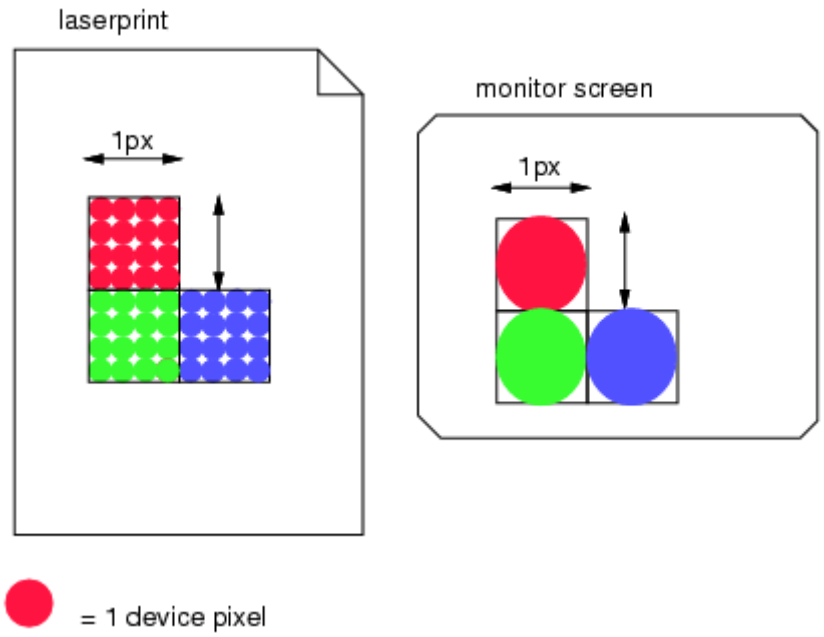

Figure 2 *Showing that more device pixels (dots) are needed to cover a 1px by 1px area on a high-resolution device than on a lower-resolution one (of the same approximate viewing distance)*

A ***device pixel*** is the smallest unit of area on the device output capable of displaying its full range of colors. For typical color screens, it's a square or somewhat rectangular region containing a red,

green, and blue subpixel. Many non-traditional outputs exist that can blur this definition, such as by displaying some colors at higher resolutions. Such devices still expose some equivalent notion of "device pixel", however.

6. Other Quantities

6.1. Angle Units: the <angle> type and 'deg', 'grad', 'rad', 'turn' units

Angle values are <dimension>s denoted by ***'<angle>'***. The angle unit identifiers are:

'deg'
: Degrees. There are 360 degrees in a full circle.

'grad'
: Gradians, also known as "gons" or "grades". There are 400 gradians in a full circle.

'rad'
: Radians. There are 2π radians in a full circle.

'turn'
: Turns. There is 1 turn in a full circle.

For example, a right angle is '90deg' or '100grad' or '0.25turn' or approximately '1.57rad'.

All <angle> units are compatible, and 'deg' is their canonical unit.

> By convention, when an angle denotes a direction in CSS, it is typically interpreted as a ***bearing angle***, where 0deg is "up" or "north" on the screen, and larger angles are more clockwise (so 90deg is "right" or "east").
>
> For example, in the 'linear-gradient()' function, the <angle> that determines the direction of the gradient is interpreted as a bearing angle.

> Note: For legacy reasons, some uses of <angle> allow a bare '0' to mean '0deg'. This is not true in general, however, and will not occur in future uses of the <angle> type.

6.2. Duration Units: the <time> type and 's', 'ms' units

Time values are dimensions denoted by ***'<time>'***. The time unit identifiers are:

's'
: Seconds.

'ms'

Milliseconds. There are 1000 milliseconds in a second.

All <time> units are compatible, and 's' is their canonical unit.

Properties may restrict the time value to some range. If the value is outside the allowed range, the declaration is invalid and must be ignored.

§ 6.3. Frequency Units: the <frequency> type and 'Hz', 'kHz' units

Frequency values are dimensions denoted by ***'<frequency>'***. The frequency unit identifiers are:

'Hz'
Hertz. It represents the number of occurrences per second.

'kHz'
KiloHertz. A kiloHertz is 1000 Hertz.

For example, when representing sound pitches, 200Hz (or 200hz) is a bass sound, and 6kHz (or 6khz) is a treble sound.

All <frequency> units are compatible, and 'hz' is their canonical unit.

> Note: Units are ASCII case-insensitive and serialize as lower case, for example 1Hz serializes as 1hz.

§ 6.4. Resolution Units: the <resolution> type and 'dpi', 'dpcm', 'dppx' units

Resolution units are dimensions denoted by ***'<resolution>'***. The resolution unit identifiers are:

'dpi'
Dots per inch.

'dpcm'
Dots per centimeter.

'dppx'
Dots per 'px' unit.

The <resolution> unit represents the size of a single "dot" in a graphical representation by indicating how many of these dots fit in a CSS 'in', 'cm', or 'px'. For uses, see e.g. the 'resolution' media query in [MEDIAQ] or the 'image-resolution' property defined in [CSS3-IMAGES].

All <resolution> units are compatible, and 'dppx' is their canonical unit.

Note that due to the 1:96 fixed ratio of CSS 'in' to CSS 'px', '1dppx' is equivalent to '96dpi'. This corresponds to the default resolution of images displayed in CSS: see 'image-resolution'.

EXAMPLE 15

The following @media rule uses Media Queries [MEDIAQ] to assign some special style rules to devices that use two or more device pixels per CSS 'px' unit:

```
@media (min-resolution: 2dppx) { ... }
```

7. Data Types Defined Elsewhere

Some data types are defined in their own modules. This example talks about some of the most common ones used across several specifications.

7.1. Colors: the <color> type

The <color> data type is defined in [CSS3COLOR]. UAs that support CSS Color Level 3 or its successor must interpret <color> as defined therein.

7.2. Images: the <image> type

The <image> data type is defined in [CSS3-IMAGES]. UAs that support CSS Images Level 3 or its successor must interpret <image> as defined therein. UAs that do not yet support CSS Images Level 3 must interpret <image> as <url>.

7.3. 2D Positioning: the <position> type

The ***'<position>'*** value specifies the position of a object area (e.g. background image) inside a positioning area (e.g. background positioning area). It is interpreted as specified for 'background-position'. [CSS3-BACKGROUND]

```
<position> = [
  [ left | center | right ] || [ top | center | bottom ]
|
  [ left | center | right | <length-percentage> ]
  [ top | center | bottom | <length-percentage> ]?
|
```

```
  [ [ left | right ] <length-percentage> ] &&
  [ [ top | bottom ] <length-percentage> ]
]
```

> Note: The 'background-position' property also accepts a three-value syntax. This has been disallowed generically because it creates parsing ambiguities when combined with other length or percentage components in a property value.

The canonical order when serializing is the horizontal component followed by the vertical component.

When specified in a grammar alongside other keywords, <length>s, or <percentage>s, <position> is *greedily* parsed; it consumes as many components as possible.

> EXAMPLE 16
> For example, 'transform-origin' defines a 3D position as (effectively) "<position> <length>?". A value such as 'left 50px' will be parsed as a 2-value <position>, with an omitted z-component; on the other hand, a value such as 'top 50px' will be parsed as a single-value <position> followed by a <length>.

§ 8. Functional Notations

A ***functional notation*** is a type of component value that can represent more complex types or invoke special processing. The syntax starts with the name of the function immediately followed by a left parenthesis (i.e. a <function-token>) followed by the argument(s) to the notation followed by a right parenthesis. White space is allowed, but optional, immediately inside the parentheses. Functions can take multiple arguments, which are formatted similarly to a CSS property value.

Some legacy functional notations, such as 'rgba()', use commas unnecessarily, but generally commas are only used to separate items in a list, or pieces of a grammar that would be ambiguous otherwise. If a comma is used to separate arguments, white space is optional before and after the comma.

> EXAMPLE 17
>
> ```
> background: url(http://www.example.org/image);
> color: rgb(100, 200, 50 );
> content: counter(list-item) ". ";
> width: calc(50% - 2em);
> ```

§ 8.1. Mathematical Expressions: 'calc()'

The ***'calc()'*** function allows a numeric CSS component value to be written as a mathematical expression using addition ('+'), subtraction ('-'), multiplication ('*'), and/or division ('/').

The 'calc()' expression represents the result of the mathematical calculation it contains, which is evaluated using standard operator precedence rules ('*' and '/' bind tighter than '+' and '-', and operators are otherwise evaluated left-to-right).

It can be used wherever <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> values are allowed. Components of a 'calc()' expression can be literal values or or 'calc()' expressions.

EXAMPLE 18

```
section {
  float: left;
  margin: 1em; border: solid 1px;
  width: calc(100%/3 - 2*1em - 2*1px);
}
```

EXAMPLE 19

```
p {
  margin: calc(1rem - 2px) calc(1rem - 1px);
}
```

EXAMPLE 20

The following sets the 'font-size' so that exactly 40em fits within the viewport, ensuring that roughly the same amount of text always fills the screen no matter the screen size.

```
:root {
  font-size: calc(100vw / 40);
}
```

If the rest of the design is specified using the 'rem' unit, the entire layout will scale to match the viewport width.

EXAMPLE 21

The following example stacks two background images, with the first perfectly centered and the second offset 20px to the bottom and to the left of the first.

```
.foo {
  background: url(top.png), url(bottom.png);
  background-repeat: no-repeat;
  background-position: 50% 50%, calc(50% + 20px) calc(50% + 20px);
}
```

EXAMPLE 22

This example shows how to place color-stops on a gradient an equal distance from either end.

```
.foo {
  background-image: linear-gradient(to right, silver,
                                                white 50px,
                                                white calc(100% - 50px),
                                                silver);
}
```

§ 8.1.1. Syntax

The syntax of a 'calc()' function is:

```
<calc()> = calc( <calc-sum> )
<calc-sum> = <calc-product> [ [ '+' | '-' ] <calc-product> ]*
<calc-product> = <calc-value> [ '*' <calc-value> | '/' <calc-number-value> ]*
<calc-value> = <number> | <dimension> | <percentage> | ( <calc-sum> )
<calc-number-sum> = <calc-number-product> [ [ '+' | '-' ] <calc-number-product>
<calc-number-product> = <calc-number-value> [ '*' <calc-number-value> | '/' <ca
<calc-number-value> = <number> | ( <calc-number-sum> )
```

In addition, white space is required on both sides of the '+' and '-' operators. (The '*' and '/' operators can be used without white space around them.)

UAs must support 'calc()' expressions of at least 20 terms, where each `NUMBER`, `DIMENSION`, or `PERCENTAGE` is a term. If a 'calc()' expression contains more than the supported number of terms, it must be treated as if it were invalid.

§ 8.1.2. Type Checking

A math expression has a ***resolved type***, which is one of <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer>. The resolved type must be valid for where the expression is placed; otherwise, the expression is invalid. The resolved type of the expression is determined by the types of the values it contains. <number-token>s are of type <number> or <integer>. A <dimension-token>'s type is given by its unit ('cm' is <length>, 'deg' is <angle>, etc.).

> Note: Because <number-token>s are always interpreted as <number>s or <integer>s, "unitless 0" <length>s aren't supported in 'calc()'. That is, 'width: calc(0 + 5px);' is invalid, even though both 'width: 0;' and 'width: 5px;' are valid.

If percentages are accepted in the context in which the expression is placed, and they are defined to be relative to another type besides <number>, a <percentage-token> is treated as that type. For example, in the 'width' property, percentages have the <length> type. A percentage only has the <percentage> type if in that context <percentage> values are not used-value compatible with any other type. If percentages are not normally allowed in place of the 'calc()', then a 'calc()' expression containing percentages is invalid in that context.

> Note: Note that <percentage>s relative to <number>s, such as in 'opacity', are not allowed in 'calc()'. Allowing this would cause significant problems with "unit algebra" (allowing multiplication/division of <dimension>s), and in every case so far, doesn't provide any new functionality. (For example, 'opacity: 25%' is identical to 'opacity: .25'; it's just a trivial syntax transform.)

> Note: Although there are a few properties in which a bare <number> becomes a <length> at used-value time (specifically, 'line-height' and 'tab-size'), <number>s never become "length-like" in 'calc()'. They always stay as <number>s.

Operators form sub-expressions, which gain types based on their arguments. To make expressions simpler, operators have restrictions on the types they accept. At each operator, the types of the left and right argument are checked for these restrictions. If compatible, the type resolves as described below (the following ignores precedence rules on the operators for simplicity):

- At '+' or '-', check that both sides have the same type, or that one side is a <number> and the other is an <integer>. If both sides are the same type, resolve to that type. If one side is a <number> and the other is an <integer>, resolve to <number>.
- At '*', check that at least one side is <number>. If both sides are <integer>, resolve to <integer>. Otherwise, resolve to the type of the other side.
- At '/', check that the right side is <number>. If the left side is <integer>, resolve to <number>. Otherwise, resolve to the type of the left side.

If an operator does not pass the above checks, the expression is invalid. Also, division by zero is invalid. This includes both dividing by the literal number zero, as well as any numeric expression that evaluates to zero (as purely-numeric expressions can be evaluated without any additional information at parse time).

> Note: Algebraic simplifications do not affect the validity of the 'calc()' expression or its resolved type. For example, 'calc(5px - 5px + 10s)' and 'calc(0 * 5px + 10s)' are both invalid due to the attempt to add a length and a time.

§ 8.1.3. Computed Value

The computed value of a 'calc()' expression is the expression with all components computed.

Where percentages are not resolved at computed-value time, they are not resolved in 'calc()' expressions, e.g. 'calc(100% - 100% + 1em)' resolves to 'calc(1em + 0%)', not to '1em'. If there are special rules for computing percentages in a value (e.g. the 'height' property), they apply whenever a 'calc()' expression contains percentages.

> EXAMPLE 23
> For example, whereas 'font-size' computes percentage values at computed value time so that font-relative length units can be computed, 'background-position' has layout-dependent behavior for percentage values, and thus does not resolve percentages until used-value time.
>
> Due to this, 'background-position' computation preserves the percentage in a 'calc()' whereas 'font-size' will compute such expressions directly into a length.

Given the complexities of width and height calculations on table cells and table elements, math expressions involving percentages for widths and heights on table columns, table column groups, table rows, table row groups, and table cells in both auto and fixed layout tables MAY be treated as if 'auto' had been specified.

§ 8.1.4. Range Checking

Parse-time range-checking of values is not performed within 'calc()', and therefore out-of-range values do not cause the declaration to become invalid. However, the value resulting from an expression must be clamped to the range allowed in the target context. Clamping is performed on computed values to the extent possible, and also on used values if computation was unable to sufficiently simplify the expression to allow range-checking. (Clamping is not performed on specified values.)

Note: This requires all contexts accepting 'calc()' to define their allowable values as a closed (not open) interval.

EXAMPLE 24
Since widths smaller than 0px are not allowed, these three declarations are equivalent:

```
width: calc(5px - 10px);
width: calc(-5px);
width: 0px;
```

Note however that 'width: -5px' is not equivalent to 'width: calc(-5px)'! Out-of-range values *outside* 'calc()' are syntactically invalid, and cause the entire declaration to be dropped.

8.1.5. Serialization

The serialization of 'calc()' values is undefined in this level.

Appendix A: IANA Considerations

Registration for the `about:invalid` URL scheme

This sections defines and registers the `about:invalid` URL, in accordance with the registration procedure defined in [RFC6694].

The official record of this registration can be found at http://www.iana.org/assignments/about-uri-tokens/about-uri-tokens.xhtml.

Registered Token	`invalid`
Intended Usage	The `about:invalid` URL references a non-existent document with a generic error condition. It can be used when a URL is necessary, but the default value shouldn't be resolvable as any type of document.
Contact/Change controller	CSS WG <www-style@w3.org> (on behalf of W3C)
Specification	CSS Values and Units Module Level 3

§ Acknowledgments

Comments and suggestions from Giovanni Campagna, Christoph Päper, Keith Rarick, Alex Mogilevsky, Ian Hickson, David Baron, Edward Welbourne, Boris Zbarsky, Björn Höhrmann and Michael Day improved this module.

§ Changes

Changes since the 6 June 2019 Candidate Recommendation are:

- Dropped the 'attr()' function, since it was punted to Level 5.
- Specified serialization of empty urls to be `url("")`. (Issue 6447)
- Clarified that the font-relative lengths are calculated without text shaping. (Issue 5498)
- Added a definition for device pixel. (Issue 7287)
- Cleaned up interaction of this specification and [CSS-CASCADE-3] in defining the CSS-wide keywords. (Issue 7439)
- Removed definition of <length-number>, since it is impossible to combine them with 'calc()'. (Issue 2789)
- Fixed definition of numbers to allow decimals in combination with scientific notation, as originally intended and as defined in [CSS-SYNTAX-3]. (Issue 7248)
- Editorial improvements.

Changes since the 31 January 2019 Candidate Recommendation are:

- Added the new bracketed range notation to the CSS value definition syntax. This has no normative implications on implementations, just allows more routine annotation of ranges in future CSS specifications. (Issue 355)

Changes since the 14 August 2018 Candidate Recommendation are:

- Defined <'property'> syntax to refer to the property without any top-level #-multiplier, to make the notation usable with common list-valued property patterns. (Issue 3146)
- Clarified that numeric values outside the allowed range are not ignored if a more specific spec defines other handling. (Issue 3270)

  > Properties may restrict numeric values to some range. If the value is outside the allowed range, unless otherwise specified, the declaration is invalid and must be ignored.

- Fixed some 'background-position' examples to be less confusing. (Issue 3482)

A Disposition of Comments is available.

Changes since the 29 September 2016 Candidate Recommendation are:

- Removed consideration of scrollbars in computing viewport units due to lack of implementations. (Issue 15)
- Inlined the <position> definition and dropped the three-value syntaxes to allow for unambiguous combination in complex grammars. This effectively removes that syntax from 'object-position', but allows <position> to be re-used e.g. in [CSS-TRANSFORMS-1] for 3D positions. (See discussion.)
- Reverted previous change to allow zero angles to drop their unit; this will instead be special-cased where needed for backwards-compatibility. (See discussion)
- Defined that range checking, and any resulting clamping, of 'calc()' values is performed both at computed time and at used time. (Issue #434)
- Fixed grammar error that disallowed numeric expressions as denominators in 'calc()'. (Issue 12)
- Defined handling of font-relative units outside the context of an element. (Issue 9)
- Defined that '0' parses as <number> if it's ambiguous whether it's a <number> or a <length>. (Issue 489)
- Defined empty 'url()'s to refer to an invalid URL, rather than resolving to the URL of the style sheet. (Issue 2211)
- Removed (unused) ability for percentages to be treated as a <number> type in 'calc()'. (Issue 1463)
- Clarified that high-resolution screens should anchor on device pixels, not physical units. (Issue 8)
- Clarified definition of 'url()' to normatively say that it accepts unquoted syntax.
- Defined that fragment-only 'url()' are specially handled to always be page-local links, regardless of base-url shenanigans. (See § 3.4.1.1 Fragment URLs.)
- Defined attr() parsing in terms of the Syntax spec, not CSS2.1 grammar.

A Disposition of Comments is available.

Changes since the 11 June 2015 Candidate Recommendation are:

- Dropped `toggle()` for lack of implementations.
- Allow zero angles to be represented as '0'. (Change due to Web-compatibility constraints in transform and gradient syntaxes.)
- Defined special handling for fragment URLs.

- Defined an empty <url> resolves to an invalid resource.
- Defined compatible units and canonical units for serialization.
- Defined case-sensitivity of 'url()' attribute argument to match attribute selectors.
- Added definition of <ident> notation to definition of identifiers.
- Added <length-percentage> as a shorthand for <length> | <percentage>, along with equivalent productions for angles, numbers, times, and frequencies.
- Allowed <percentage>s inside 'calc()' to resolve as their own type, if they occur in some (as yet theoretical) context where they are not compatible with any other type.
- Various clarifications and editorial improvements.

Changes since the 30 July 2013 Candidate Recommendation are:

- Specified that, in the absence of font information, '1ch' equal '.5em'.
- Added 'Q' unit.
- Relaxed unnecessary restrictions on <custom-ident>. Require specs referencing it to be clear about excluded keywords, because the new rule isn't as simple.
- Clarified relative URL resolution for embedded style sheets.
- Clarified {*A*} variant of {*A*,*B*} notation.
- Added notation for restricting the length of comma-separated lists specified with the '#' notation.
- Clarified handling of `toggle()` when used in shorthand declarations.
- Clarified that stringing together reorderable combinations allows interleaving.
- Changed syntax references from the 2.1 grammar to the Syntax spec.

A Disposition of Comments is available.

Changes since the 28 August 2012 Candidate Recommendation are:

- Corrected `wqname` in the 'attr()' syntax to `qname`
- Made undefined namespace prefixes in 'attr()' invalidate the function.
- Per WG resolution, made viewport-percentage units respect scrollbars on the viewport unless 'overflow' is 'auto' (in which case they ignore the presence of scrollbars).
- Deferred exact definition of viewport-percentage units in paged media to CSS Paged Media.
- Added back the <custom-ident> term as a convenience notation, so that other specs can refer to it.

Changes since the 4 April 2013 Candidate Recommendation are:

- Noted that the list of CSS-wide keywords may be expanded by other specs.
- Clarified definition of 'ex' to refer to the "first available font".
- Specified that 'attr()' with `string` or `url` types doesn't reparse the attribute contents, just takes the value literally as the value of a <string>.

§ Security Considerations

This specification presents no new security considerations.

This specification defines the 'url()' function (<url>), which allows CSS to make network requests. Depending on what features they are used in, these can potentially expose whether or not the user has access to resources on a network, and expose information about their contents (such as the rules within a style sheet, the size of an image, the metrics of a font). They can also allow exfiltrating data via URL.

§ Privacy Considerations

This specification introduces units that expose the user's screen size (the viewport-percentage lengths), default font size, and potentially some information about which fonts are available on the user's system (the font-relative lengths).

This specification defines the 'url()' function (<url>), which allows CSS to make network requests. Depending on what features they are used in, these can potentially expose whether or not the user has access to resources on a network, and expose information about their contents (such as the rules within a style sheet, the size of an image, the metrics of a font). They can also allow exfiltrating data via URL.

§ 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 25
>
> 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.

§ 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

rem, in § 5.1.1

<resolution>, in § 6.4

resolved type, in § 8.1.2

s, in § 6.2

<string>, in § 3.3

textual data types, in § 3

<time>, in § 6.2

<time-percentage>, in § 4.6

turn, in § 6.1

<url>, in § 3.4

URL, in § 3.4

url(), in § 3.4

<url-modifier>, in § 3.4.3

value definition syntax, in § 2

vh, in § 5.1.2

viewport-percentage lengths, in § 5.1.2

visual angle unit, in § 5.2

vmax, in § 5.1.2

vmin, in § 5.1.2

vw, in § 5.1.2

§ Terms defined by reference

[CSS-ANIMATIONS-1] defines the following terms:

- animation
- animation-name
- animation-timing-function

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

- padding-top

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

- orphans

[CSS-CASCADE-5] defines the following terms:

- @import
- actual value
- computed value
- inherit
- initial
- specified value
- unset
- used value

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

- <color>
- opacity
- rgba()

[CSS-COLOR-5] defines the following terms:

- hsl()

[CSS-COUNTER-STYLES-3] defines the following terms:

- disc

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

- containing block

[CSS-EASING-1] defines the following terms:

- <easing-function>
- ease-in
- ease-out

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

- font
- font-family
- font-size

[CSS-GRID-2] defines the following terms:

- fr

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

- image-resolution

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

- auto
- overflow

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

- auto
- width

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

- <dimension-token>
- <function-token>
- <ident-token>
- <number-token>
- <percentage-token>
- <string-token>
- <url-token>
- component value
- consume a url token

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

- center
- tab-size
- text-align

[CSS-TEXT-DECOR-3] defines the following terms:

- text-decoration

[CSS-TRANSFORMS-1] defines the following terms:

- transform-origin

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

- default
- outline-color

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

- attr()

[CSS-WRITING-MODES-4] defines the following terms:

- text-orientation
- upright
- vertical-lr
- vertical-rl
- writing-mode

[CSS21] defines the following terms:

- <border-width>
- border-collapse
- line-height

[CSS3-BACKGROUND] defines the following terms:

- background
- background-attachment
- background-position
- border-color
- border-width
- box-shadow

[CSS3-IMAGES] defines the following terms:

- <image>
- linear-gradient()
- object-position

[HTML] defines the following terms:

- base
- pushState(data, unused)

[INFRA] defines the following terms:

- ascii case-insensitive
- string

[MEDIAQUERIES-5] defines the following terms:

media query

paged media

[SELECTORS-3] defines the following terms:

*

[SELECTORS-4] defines the following terms:

+

§ References

§ Normative References

[CSS-CASCADE-5]

Elika Etemad; Miriam Suzanne; Tab Atkins Jr.. *CSS Cascading and Inheritance Level 5*. 13 January 2022. CR. URL: https://www.w3.org/TR/css-cascade-5/

[CSS-COLOR-4]

Tab Atkins Jr.; Chris Lilley; Lea Verou. *CSS Color Module Level 4*. 1 November 2022. CR. URL: https://www.w3.org/TR/css-color-4/

[CSS-COUNTER-STYLES-3]

Tab Atkins Jr.. *CSS Counter Styles Level 3*. 27 July 2021. CR. URL: https://www.w3.org/TR/css-counter-styles-3/

[CSS-DISPLAY-3]

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

[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-IMAGES-4]

Tab Atkins Jr.; Elika Etemad; Lea Verou. *CSS Image Values and Replaced Content Module Level 4*. 13 April 2017. WD. URL: https://www.w3.org/TR/css-images-4/

[CSS-OVERFLOW-3]

David Baron; Elika Etemad; Florian Rivoal. *CSS Overflow Module Level 3*. 23 December 2021. WD. URL: https://www.w3.org/TR/css-overflow-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-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-UI-3]

Tantek Çelik; Florian Rivoal. *CSS Basic User Interface Module Level 3 (CSS3 UI)*. 21 June 2018. REC. URL: https://www.w3.org/TR/css-ui-3/

[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-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/

[CSS21]

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

[CSS3-BACKGROUND]

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

[CSS3-FONTS]

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

[CSS3-IMAGES]

Tab Atkins Jr.; Elika Etemad; Lea Verou. *CSS Images Module Level 3*. 17 December 2020. CR. URL: https://www.w3.org/TR/css-images-3/

[CSS3COLOR]

Tantek Çelik; Chris Lilley; David Baron. *CSS Color Module Level 3*. 18 January 2022. REC. URL: https://www.w3.org/TR/css-color-3/

[CSS3PAGE]

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

[CSSOM]

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

[INFRA]

Anne van Kesteren; Domenic Denicola. *Infra Standard*. Living Standard. URL: https://infra.spec.whatwg.org/

[MEDIAQUERIES-5]

Dean Jackson; et al. *Media Queries Level 5*. 18 December 2021. WD. URL: https://www.w3.org/TR/mediaqueries-5/

[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

[SELECTORS-3]

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*. 7 May 2022. 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-ANIMATIONS-1]

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

[CSS-BOX-4]

Elika Etemad. *CSS Box Model Module Level 4*. 3 November 2022. WD. URL: https://www.w3.org/TR/css-box-4/

[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-CASCADE-3]

Elika Etemad; Tab Atkins Jr.. *CSS Cascading and Inheritance Level 3*. 11 February 2021. REC. URL: https://www.w3.org/TR/css-cascade-3/

[CSS-COLOR-5]

Chris Lilley; et al. *CSS Color Module Level 5*. 28 June 2022. WD. URL: https://www.w3.org/TR/css-color-5/

[CSS-EASING-1]

Brian Birtles; et al. *CSS Easing Functions Level 1*. 1 April 2021. CR. URL: https://www.w3.org/TR/css-easing-1/

[CSS-GRID-1]

Tab Atkins Jr.; et al. *CSS Grid Layout Module Level 1*. 18 December 2020. CR. URL: https://www.w3.org/TR/css-grid-1/

[CSS-GRID-2]

Tab Atkins Jr.; Elika Etemad; Rossen Atanassov. *CSS Grid Layout Module Level 2*. 18

December 2020. CR. URL: https://www.w3.org/TR/css-grid-2/

[CSS-TEXT-3]

Elika Etemad; Koji Ishii; Florian Rivoal. *CSS Text Module Level 3*. 5 May 2022. CR. URL: https://www.w3.org/TR/css-text-3/

[CSS-TEXT-DECOR-3]

Elika Etemad; Koji Ishii. *CSS Text Decoration Module Level 3*. 5 May 2022. CR. URL: https://www.w3.org/TR/css-text-decor-3/

[CSS-TRANSFORMS-1]

Simon Fraser; et al. *CSS Transforms Module Level 1*. 14 February 2019. CR. URL: https://www.w3.org/TR/css-transforms-1/

[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/

[RFC6694]

S. Moonesamy, Ed.. *The "about" URI Scheme*. August 2012. Informational. URL: https://www.rfc-editor.org/rfc/rfc6694