# Media Queries Level 4

Editor’s Draft, 30 July 2020

▶ **Specification Metadata**

## Abstract

Media Queries allow authors to test and query values or features of the user agent or display device, independent of the document being rendered. They are used in the CSS @media rule to conditionally apply styles to a document, and in various other contexts and languages, such as HTML and JavaScript.

Media Queries Level 4 describes the mechanism and syntax of media queries, media types, and media features. It extends and supersedes the features defined in Media Queries Level 3.

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 is a public copy of the editors’ draft. It is provided for discussion only and may change at any moment. Its publication here does not imply endorsement of its contents by W3C. Don’t cite this document other than as work in progress.

Please send feedback by filing issues in GitHub (preferred), including the spec code “mediaqueries” in the title, like this: “[mediaqueries] *…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.

There is currently no preliminary interoperability or implementation report.

The following features are at-risk, and may be dropped during the CR period:

- The ‘update’ media feature

“At-risk” is a W3C Process term-of-art, and does not necessarily imply that the feature is in danger of being dropped or delayed. It means that the WG believes the feature may have difficulty being interoperably implemented in a timely manner, and marking it as such allows the WG to drop the feature if necessary when transitioning to the Proposed Rec stage, without having to publish a new Candidate Rec without the feature first.

## Table of Contents

## § 1. Introduction

*This section is not normative.*

In 1997, HTML4 [HTML401] defined a mechanism to support media-dependent style sheets, tailored for different media types. For example, a document may use different style sheets for screen and for print. In HTML, this can be written as:

EXAMPLE 1

```
<link rel="stylesheet" type="text/css" media="screen" href="style.css">
<link rel="stylesheet" type="text/css" media="print" href="print.css">
```

CSS adapted and extended this functionality with its '@media' and '@import' rules, adding the ability to query the value of individual features:

EXAMPLE 2

Inside a CSS style sheet, one can declare that sections apply to certain media types:

```
@media screen {
  * { font-family: sans-serif }
}
```

Similarly, stylesheets can be conditionally imported based on media queries:

```
@import "print-styles.css" print;
```

Media queries can be used with HTML, XHTML, XML [xml-stylesheet] and the @import and @media rules of CSS.

EXAMPLE 3

Here is the same example written in HTML, XHTML, XML, @import and @media:

```
<link media="screen and (color), projection and (color)"
      rel="stylesheet" href="example.css">

<link media="screen and (color), projection and (color)"
      rel="stylesheet" href="example.css" />

<?xml-stylesheet media="screen and (color), projection and (color)"
                 rel="stylesheet" href="example.css" ?>

@import url(example.css) screen and (color), projection and (color);

@media screen and (color), projection and (color) { … }
```

## § 1.1. Module interactions

This module replaces and extends the Media Queries, Media Type and Media Features defined in [CSS2] sections 7 and in [MEDIAQUERIES-3].

## § 1.2. Values

Value types not defined in this specification, such as <integer>, <number> or <resolution>, are defined in [CSS-VALUES-3]. Other CSS modules may expand the definitions of these value types.

## § 1.3. Units

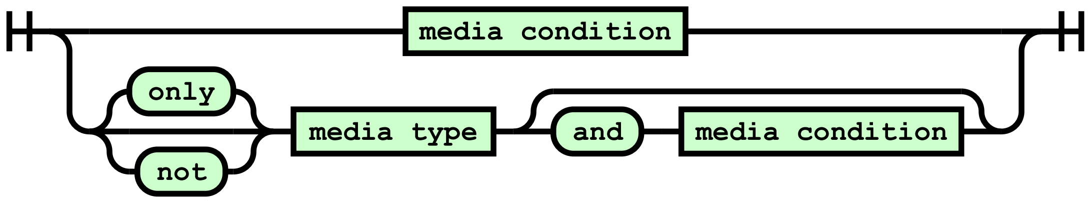
The units used in media queries are the same as in other parts of CSS, as defined in [CSS-VALUES-3]. For example, the pixel unit represents CSS pixels and not physical pixels.

Relative length units in media queries are based on the initial value, which means that units are never based on results of declarations. For example, in HTML, the 'em' unit is relative to the initial value of 'font-size', defined by the user agent or the user's preferences, not any styling on the page.

# § 2. Media Queries

A ***media query*** is a method of testing certain aspects of the user agent or device that the document is being displayed in. Media queries are (almost) always independent of the contents of the document, its styling, or any other internal aspect; they're only dependent on "external" information unless another feature explicitly specifies that it affects the resolution of Media Queries.

The syntax of a media query consists of an optional media query modifier, an optional media type, and zero or more media features:



A media query is a logical expression that is either true or false. A media query is true if:

- the media type, if specified, matches the media type of the device where the user agent is running, and
- the media condition is true.

Statements regarding media queries in this section assume the syntax section is followed. Media queries that do not conform to the syntax are discussed in § 3.2 Error Handling. I.e. the syntax takes precedence over requirements in this section.

EXAMPLE 4

Here is a simple example written in HTML:

```
<link rel="stylesheet" media="screen and (color)" href="example.css" />
```

This example expresses that a certain style sheet (`example.css`) applies to devices of a certain media type ('screen') with certain feature (it must be a color screen).

Here is the same media query written in an @import-rule in CSS:

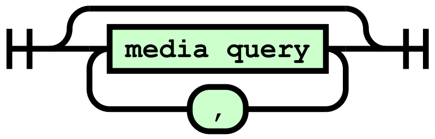
```
@import url(example.css) screen and (color);
```

User agents must re-evaluate media queries in response to changes in the user environment that they're aware of, for example if the device is tiled from landscape to portrait orientation, and change the behavior of any constructs dependent on those media queries accordingly.

Unless another feature explicitly specifies that it affects the resolution of Media Queries, it is never necessary to apply a style sheet in order to evaluate expressions.

## § 2.1. Combining Media Queries

Several media queries can be combined into a comma-separated ***media query list***.



A media query list is true if *any* of its component media queries are true, and false only if *all* of its component media queries are false.

EXAMPLE 5

For example, the following media query list is true if either the media type is ‘screen’ and it’s a color device, **or** the media type is ‘projection’ and it’s a color device:

```
@media screen and (color), projection and (color) { … }
```

An empty media query list evaluates to true.

EXAMPLE 6

For example, these are equivalent:

```
@media all { … }
@media { … }
```

## § 2.2. Media Query Modifiers

A media query may optionally be prefixed by a single ***media query modifier***, which is a single keyword which alters the meaning of the following media query.

### § 2.2.1. Negating a Media Query: the ‘not’ keyword

An individual media query can have its result negated by prefixing it with the keyword ***‘not’***. If the media query would normally evaluate to true, prefixing it with ‘not’ makes it evaluate to false, and vice versa.

EXAMPLE 7

For example, the following will apply to everything except color-capable screens. Note that the entire media query is negated, not just the media type.

```
<link rel="stylesheet" media="not screen and (color)" href="example.css" />
```

### § 2.2.2. Hiding a Media Query From Legacy User Agents: the ‘only’ keyword

The concept of media queries originates from HTML4 [HTML401]. That specification only defined media types, but had a forward-compatible syntax that accommodated the addition of future concepts like media features: it would consume the characters of a media query up to the first non-alphanumeric character, and interpret that as a media type, ignoring the rest. For example, the media query ‘screen and (color)’ would be truncated to just ‘screen’.

Unfortunately, this means that legacy user agents using this error-handling behavior will ignore any media features in a media query, even if they’re far more important than the media type in the query. This can result in styles accidentally being applied in inappropriate situations.

To hide these media queries from legacy user agents, the media query can be prefixed with the keyword ***‘only’***. The ‘only’ keyword **has no effect** on the media query’s result, but will cause the media query to be parsed by legacy user agents as specifying the unknown media type “only”, and thus be ignored.

EXAMPLE 8

In this example, the stylesheet specified by the `<link>` element will not be used by legacy user agents, even if they would normally match the ‘screen’ media type.

```
<link rel="stylesheet" media="only screen and (color)" href="example.css" />
```

Note: Note that the ‘only’ keyword can only be used before a media type. A media query consisting only of media features, or one with another media query modifier like ‘not’, will be treated as false by legacy user agents automatically.

Note: At the time of publishing this specification, such legacy user agents are extremely rare, and so using the ‘only’ modifier is rarely, if ever, necessary.

## § 2.3. Media Types

A ***media type*** is a broad category of user-agent devices on which a document may be displayed. The original set of media types were defined in HTML4, for the `media` attribute on `<link>` elements.

Unfortunately, media types have proven insufficient as a way of discriminating between devices with different styling needs. Some categories which were originally quite distinct, such as ‘screen’ and ‘handheld’, have blended significantly in the years since their invention. Others, such as ‘tty’ or ‘tv’, expose useful differences from the norm of a full-featured computer monitor, and so are potentially useful to target with different styling, but the definition of media types as mutually exclusive makes it difficult to use them in a reasonable manner; instead, their exclusive aspects are better expressed as media features such as ‘grid’ or ‘scan’.

As such, the following media types are defined for use in media queries:

***‘all’***
Matches all devices.

***‘print’***
Matches printers, and devices intended to reproduce a printed display, such as a web browser showing a document in “Print Preview”.

***‘screen’***
Matches all devices that aren’t matched by ‘print’.

In addition, the following **deprecated** media types are defined. Authors must not use these media types; instead, it is recommended that they select appropriate media features that better represent the aspect of the device that they are attempting to style against.

User agents must recognize the following media types as valid, but must make them match nothing.

- ***‘tty’***
- ***‘tv’***
- ***‘projection’***
- ***‘handheld’***
- ***‘braille’***
- ***‘embossed’***

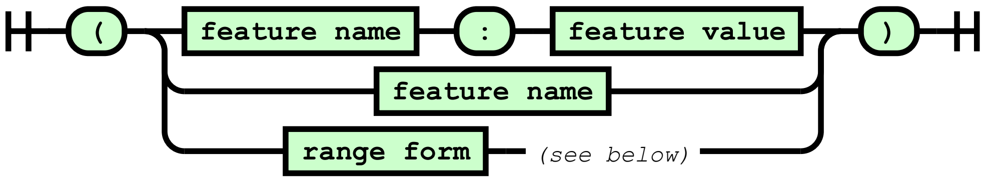
- ***‘aural’***
- ***‘speech’***

Note: It is expected that all of the media types will also be deprecated in time, as appropriate media features are defined which capture their important differences.

## § 2.4. Media Features

A ***media feature*** is a more fine-grained test than media types, testing a single, specific feature of the user agent or display device.

Syntactically, media features resemble CSS properties: they consist of a feature name, a colon, and a value to test for. They may also be written in boolean form as just a feature name, or in range form with a comparison operator.



There are, however, several important differences between properties and media features:

- Properties are used to give information about how to present a document. Media features are used to describe requirements of the output device.
- Media features are always wrapped in parentheses and combined with the ‘and’ or ‘or’ keywords, like ‘(color) and (min-width: 600px)’, rather than being separated with semicolons.
- A media feature may be given with *only* its name (omitting the colon and value) to evaluate the feature in a boolean context. This is a convenient shorthand for features that have a reasonable value representing 0 or “none”. For example, ‘(color)’ is true if the ‘color’ media feature is non-zero.
- Media features with “range” type can be written in a range context, which uses standard mathematical comparison operators rather than a colon, or have their feature names prefixed with “min-” or “max-”.
- Properties sometimes accept complex values, e.g., calculations that involve several other values. Media features only accept single values: one keyword, one number, etc.

If a media feature references a concept which does not exist on the device where the UA is running (for example, speech UAs do not have a concept of "width"), the media feature must always evaluate to false.

EXAMPLE 9

The media feature 'device-aspect-ratio' only applies to visual devices. On an 'speech' device, expressions involving 'device-aspect-ratio' will therefore always be false:

```
<link media="speech and (device-aspect-ratio: 16/9)"
      rel="stylesheet" href="example.css">
```

### § 2.4.1. Media Feature Types: "range" and "discrete"

Every media feature defines its "type" as either "range" or "discrete" in its definition table.

"Discrete" media features, like 'pointer' take their values from a set. The values may be keywords or boolean numbers (0 and 1), but the common factor is that there's no intrinsic "order" to them—none of the values are "less than" or "greater than" each other.

"Range" media features like 'width', on the other hand, take their values from a range. Any two values can be compared to see which is lesser and which is greater.

The only significant difference between the two types is that "range" media features can be evaluated in a range context and accept "min-" and "max-" prefixes on their name. Doing either of these changes the meaning of the feature—rather than the media feature being true when the feature exactly matches the given value, it matches when the feature is greater than/less than/equal to the given value.

EXAMPLE 10

A "(width >= 600px)" media feature is true when the viewport's width is '600px' *or more*.

On the other hand, '(width: 600px)' by itself is only true when the viewport's width is *exactly* '600px'. If it's less or greater than '600px', it'll be false.

### § 2.4.2. Evaluating Media Features in a Boolean Context

While media features normally have a syntax similar to CSS properties, they can also be written more simply as just the feature name, like '(color)'.

When written like this, the media feature is evaluated in a ***boolean context***. If the feature would be true for any value *other than* the number ‘0’, a <dimension> with the value ‘0’, or the keyword ‘none’, the media feature evaluates to true. Otherwise, it evaluates to false.

EXAMPLE 11

Some media features are designed to be written like this.

For example, ‘update’ is typically written as ‘(update)’ to test if any kind of updating is available, or ‘not (update)’ to check for the opposite.

It can still be given an explicit value as well, with ‘(update: fast) or (update: slow)’ equal to ‘(update)’, and ‘(update: none)’ equal to ‘not (update)’.

EXAMPLE 12

Some numeric media features, like ‘width’, are rarely if ever useful to evaluate in a boolean context, as their values are almost always greater than zero. Others, like ‘color’, have meaningful zero values: ‘(color)’ is identical to ‘(color > 0)’, indicating that the device is capable of displaying color at all.

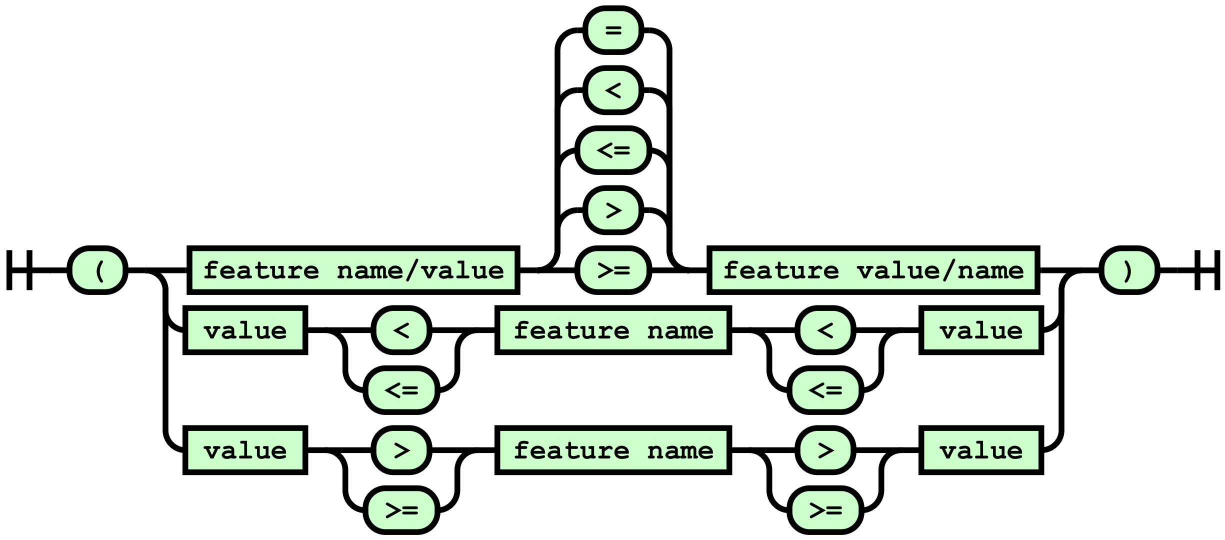
EXAMPLE 13

Only some of the media features that accept keywords are meaningful in a boolean context.

For example, ‘(pointer)’ is useful, as ‘pointer’ has a ‘none’ value to indicate there’s no pointing device at all on the device. On the other hand, ‘(scan)’ is just always true or always false (depending on whether it applies at all to the device), as there’s no value that means “false”.

## § 2.4.3. Evaluating Media Features in a Range Context

Media features with a “range” type can be alternately written in a ***range context*** that takes advantage of the fact that their values are ordered, using ordinary mathematical comparison operators:



Note: This syntax is new to Level 4 of Mediaqueries, and thus is not as widely supported at the moment as the ‘min-’/‘max-’ prefixes.

The basic form, consisting of a feature name, a comparison operator, and a value, returns true if the relationship is true.

EXAMPLE 14
For example, ‘(height > 600px)’ (or ‘(600px < height)’) returns true if the viewport height is greater than ‘600px’.

The remaining forms, with the feature name nested between two value comparisons, returns true if both comparisons are true.

EXAMPLE 15
For example, ‘(400px < width < 1000px)’ returns true if the viewport width is between ‘400px’ and ‘1000px’ (but not equal to either).

Some media features with a "range" type are said to be ***false in the negative range***. This means that negative values are valid and must be parsed, and that querying whether the media feature is equal to, less than, or less or equal than any such negative value must evaluate to false. Querying whether the media feature is greater, or greater or equal, than a negative value evaluates to true if the relationship is true.

Note: If negative values had been rejected at parse time instead, they would be treated as ‘unknown’ based on the error handling rules. However, in reality, whether a device’s ‘resolution’ is ‘-300dpi’ is not unknown, it is known to be false. Similarly, for any visual device, the ‘width’ of the targeted display area is known to be greater than ‘-200px’ The above rule reflects that, making intuition match what UAs do.

EXAMPLE 16

The following examples result in a green background on all visual devices:

```
@media not (width <= -100px) {
  body { background: green; }
}
```

```
@media (height > -100px) {
  body { background: green; }
}
```

```
@media not (resolution: -300dpi) {
  body { background: green; }
}
```

This is a behavior change compared to Media Queries Level 3 [MEDIAQUERIES-3], where negative values on these properties caused a syntax error. In level 3, syntax errors—including forbidden values—resulted in the entire media query being false, rather than the ‘unknown’ treatment defined in this level. Implementations updating from level 3 should make sure to change the handling of negative values for the relevant properties when they add support for the richer syntax defined in § 2.5 Combining Media Features, to avoid introducing unintended semantics.

### § 2.4.4. Using “min-” and “max-” Prefixes On Range Features

Rather than evaluating a “range” type media feature in a range context, as described above, the feature may be written as a normal media feature, but with a “min-” or “max-” prefix on the feature name.

This is equivalent to evaluating the feature in a range context, as follows:

- Using a “min-” prefix on a feature name is equivalent to using the “>=” operator. For example, ‘(min-height: 600px)’ is equivalent to "(height >= 600px)".
- Using a “max-” prefix on a feature name is equivalent to using the “<=” operator. For example, ‘(max-width: 40em)’ is equivalent to "(width <= 40em)".

Note: because “min-” and “max-” both equate to range comparisons that **include** the value, they may be limiting in certain situations.

EXAMPLE 17

For instance, authors trying to define different styles based on a breakpoint in the viewport width using “min-” and “max-” would generally offset the values they’re comparing, to ensure that both queries don’t evaluate to true simultaneously. Assuming the breakpoint is at 320px, authors would conceptually use:

```
@media (max-width: 320px) { /* styles for viewports <= 320px */ }
@media (min-width: 321px) { /* styles for viewports >= 321px */ }
```

While this ensures that the two sets of styles don’t apply simultaneously when the viewport width is 320px, it does not take into account the possibility of fractional viewport sizes which can occur as a result of non-integer pixel densities (e.g. on high-dpi displays or as a result of zooming/scaling). Any viewport widths that fall between 320px and 321px will result in none of the styles being applied.

One approach to work around this problem is to increase the precision of the values used for the comparison. Using the example above, changing the second comparison value to 320.01px significantly reduces the change that a viewport width on a device would fall between the cracks.

```
@media (max-width: 320px) { /* styles for viewports <= 320px */ }
@media (min-width: 320.01px) { /* styles for viewports >= 320.01px */ }
```

However, in these situations, range context queries (which are not limited to “>=” and “<=” comparisons) offer a more appropriate solution:

```
@media (width <= 320px) { /* styles for viewports <= 320px */ }
@media (width > 320px) { /* styles for viewports > 320px */ }
```

“Discrete” type properties do not accept “min-” or “max-” prefixes. Adding such a prefix to a “discrete” type media feature simply results in an unknown feature name.

EXAMPLE 18
For example, ‘(min-grid: 1)’ is invalid, because ‘grid’ is a “discrete” media feature, and so doesn’t accept the prefixes. (Even though the ‘grid’ media feature appears to be numeric, as it accepts the values ‘0’ and ‘1’.)

Attempting to evaluate a min/max prefixed media feature in a boolean context is invalid and a syntax error.

## § 2.5. Combining Media Features

Multiple media features can be combined together into a ***media condition*** using full boolean algebra (not, and, or).

- Any media feature can be negated by placing ‘not’ before it. For example, ‘not (color)’ inverts the meaning of ‘(color)’—since ‘(color)’ matches a device with any kind of color display, ‘not (color)’ matches a device *without* any kind of color display.
- Two or more media features can be chained together, such that the query is only true if *all* of the media features are true, by placing ‘and’ between them. For example, ‘(width < 600px) and (height < 600px)’ only matches devices whose screens are smaller than ‘600px’ wide in both dimensions.
- Alternately, two or more media features can be chained together, such that the query is true if *any* of the media features are true, by placing ‘or’ between them. For example, ‘(update: slow) or (hover: none)’ matches if the device is slow to update the screen (such as an e-reader) *or* the primary pointing device has no hover capability, perhaps indicating that one should use a layout that displays more information rather than compactly hiding it until the user hovers.
- Media conditions can be grouped by wrapping them in parentheses ‘()’ which can then be nested within a condition the same as a single media query. For example, ‘(not (color)) or (hover)’ is true on devices that are monochrome and/or that have hover capabilities. If one instead wanted to query for a device that was monochrome and *didn’t* have hover capabilities, it must instead be written as ‘not ((color) or (hover))’ (or, equivalently, as ‘(not (color)) and (not (hover))’).

It is *invalid* to mix ‘and’ and ‘or’ and ‘not’ at the same “level” of a media query. For example, ‘(color) and (pointer) or (hover)’ is illegal, as it’s unclear what was meant. Instead, parentheses can be used to group things using a particular joining keyword, yielding either ‘(color) and ((pointer) or (hover))’ or ‘((color) and (pointer)) or (hover)’. These two have very different meanings: if only ‘(hover)’ is true, the first one evaluates to false but the second evaluates to true.

## § 3. Syntax

Informal descriptions of the media query syntax appear in the prose and railroad diagrams in previous sections. The formal media query syntax is described in this section, with the rule/property grammar syntax defined in [CSS-SYNTAX-3] and [CSS-VALUES-3].

To parse a ***'<media-query-list>'*** production, parse a comma-separated list of component values, then parse each entry in the returned list as a <media-query>. Its value is the list of <media-query>s so produced.

> Note: This explicit definition of <media-query-list> parsing is necessary to make the error-recovery behavior of media query lists well-defined.

> Note: This definition of <media-query-list> parsing intentionally accepts an empty list.

> Note: As per [CSS-SYNTAX-3], tokens are ASCII case-insensitive.

```
<media-query> = <media-condition>
             | [ not | only ]? <media-type> [ and <media-condition-without-or> ]?
<media-type> = <ident>

<media-condition> = <media-not> | <media-in-parens> [ <media-and>* | <media-or>* ]
<media-condition-without-or> = <media-not> | <media-in-parens> <media-and>*
<media-not> = not <media-in-parens>
<media-and> = and <media-in-parens>
<media-or> = or <media-in-parens>
<media-in-parens> = ( <media-condition> ) | <media-feature> | <general-enclosed>

<media-feature> = ( [ <mf-plain> | <mf-boolean> | <mf-range> ] )
<mf-plain> = <mf-name> : <mf-value>
<mf-boolean> = <mf-name>
<mf-range> = <mf-name> <mf-comparison> <mf-value>
           | <mf-value> <mf-comparison> <mf-name>
           | <mf-value> <mf-lt> <mf-name> <mf-lt> <mf-value>
           | <mf-value> <mf-gt> <mf-name> <mf-gt> <mf-value>
<mf-name> = <ident>
<mf-value> = <number> | <dimension> | <ident> | <ratio>
<mf-lt> = '<' '='?
<mf-gt> = '>' '='?
<mf-eq> = '='
<mf-comparison> = <mf-lt> | <mf-gt> | <mf-eq>
```

```
<general-enclosed> = [ <function-token> <any-value> ) ] | ( <ident> <any-value> )
```

The <media-type> production does not include the keywords ‘only’, ‘not’, ‘and’, and ‘or’.

No whitespace is allowed between the “<” or “>” <delim-token>s and the following “=” <delim-token>, if it’s present.

Note: Whitespace is required between a ‘not’, ‘and’, or ‘or’ keyword and the following ‘(’ character, because without it that would instead parse as a <function-token>. This is not made explicitly invalid because it’s already covered by the above grammar. It’s fine to have whitespace between a ‘)’ and a following keyword, however.

When parsing the <media-in-parens> production, the <general-enclosed> branch must only be chosen if the input does not match either of the preceding branches. <general-enclosed> exists to allow for future expansion of the grammar in a reasonably compatible way.

## § 3.1. Evaluating Media Queries

Each of the major subexpression of <media-condition> or <media-condition-without-or> is associated with a boolean result, as follows:

**<media-condition>**
**<media-condition-without-or>**
The result is the result of the child subexpression.

**<media-in-parens>**
The result is the result of the child term.

**<media-not>**
The result is the negation of the <media-in-parens> term. The negation of unknown is unknown.

**<media-in-parens> <media-and>***
The result is true if the <media-in-parens> child term and all of the <media-in-parens> children of the <media-and> child terms are true, false if at least one of these <media-in-parens> terms are false, and unknown otherwise.

**<media-in-parens> <media-or>***
The result is false if the <media-in-parens> child term and all of the <media-in-parens> children of the <media-or> child terms are false, true if at least one of these <media-in-parens> terms are true, and unknown otherwise.

**<general-enclosed>**
The result is unknown.

Authors must not use <general-enclosed> in their stylesheets. It exists only for future-compatibility, so that new syntax additions do not invalidate too much of a <media-condition> in older user agents.

**<media-feature>**
The result is the result of evaluating the specified media feature.

If the result of any of the above productions is used in any context that expects a two-valued boolean, “unknown” must be converted to “false”.

Note: This means that, for example, when a media query is used in a ‘@media’ rule, if it resolves to “unknown” it’s treated as “false” and fails to match.

Media Queries use a three-value logic where terms can be “true”, “false”, or “unknown”. Specifically, it uses the Kleene 3-valued logic. In this logic, “unknown” means “either true or false, but we’re not sure which yet”.

In general, an unknown value showing up in a formula will cause the formula to be unknown as well, as substituting “true” for the unknown will give the formula a different result than substituting “false”. The only way to eliminate an unknown value is to use it in a formula that will give the same result whether the unknown is replaced with a true or false value. This occurs when you have “false AND unknown” (evaluates to false regardless) and “true OR unknown” (evaluates to true regardless).

This logic was adopted because <general-enclosed> needs to be assigned a truth value. In standard boolean logic, the only reasonable value is “false”, but this means that ‘not unknown(function)’ is true, which can be confusing and unwanted. Kleene’s 3-valued logic ensures that unknown things will prevent a media query from matching, unless their value is irrelevant to the final result.

## § 3.2. Error Handling

A media query that does not match the grammar in the previous section must be replaced by ‘not all’ during parsing.

Note: Note that a grammar mismatch does **not** wipe out an entire media query list, just the problematic media query. The parsing behavior defined above automatically recovers at the next top-level comma.

EXAMPLE 19

```
@media (example, all,), speech { /* only applicable to speech devices */ }
@media &test, speech           { /* only applicable to speech devices */ }
```

Both of the above media query lists are turned into ‘not all, speech’ during parsing, which has the same truth value as just ‘speech’.

Note that error-recovery only happens at the top-level of a media query; anything inside of an invalid parenthesized block will just get turned into ‘not all’ as a group. For example:

```
@media (example, speech { /* rules for speech devices */ }
```

Because the parenthesized block is unclosed, it will contain the entire rest of the stylesheet from that point (unless it happens to encounter an unmatched “)” character somewhere in the stylesheet), and turn the entire thing into a ‘not all’ media query.

An unknown <media-type> must be treated as not matching.

EXAMPLE 20

For example, the media query ‘unknown’ is false, as ‘unknown’ is an unknown media type.

But ‘not unknown’ is true, as the ‘not’ negates the false media type.

EXAMPLE 21

Remember that some keywords aren’t allowed as <media-type>s and cause parsing to fail entirely: the media query ‘or and (color)’ is turned into ‘not all’ during parsing, rather than just treating the ‘or’ as an unknown media type.

An unknown <mf-name> or <mf-value>, or disallowed <mf-value>, results in the value “unknown”. A <media-query> whose value is “unknown” must be replaced with ‘not all’.

EXAMPLE 22

```
<link media="screen and (max-weight: 3kg) and (color), (color)"rel="stylesheet"
```

As ‘max-weight’ is an unknown media feature, this media query list is turned into ‘not all, (color)’, which is equivalent to just ‘(color)’.

EXAMPLE 23

```
@media (min-orientation:portrait) { … }
```

The ‘orientation’ feature does not accept prefixes, so this is considered an unknown media feature, and turned into ‘not all’.

EXAMPLE 24
The media query ‘(color:20example)’ specifies an unknown value for the ‘color’ media feature and is therefore turned into ‘not all’.

Note that media queries are also subject to the parsing rules of the host language. For example, take the following CSS snippet:

```
@media test;,all { body { background:lime } }
```

The media query ‘test;,all’ is, parsed by itself, equivalent to ‘not all, all’, which is always true. However, CSS’s parsing rules cause the ‘@media’ rule, and thus the media query, to end at the semicolon. The remainder of the text is treated as a style rule with an invalid selector and contents.

## § 4. Viewport/Page Dimensions Media Features

### § 4.1. Width: the ‘width’ feature

| | |
|---|---|
| *Name:* | ***‘width’*** |
| *For:* | ‘@media’ |
| *Value:* | <length> |
| *Type:* | range |

The ‘width’ media feature describes the width of the targeted display area of the output device. For continuous media, this is the width of the viewport (as described by CSS2, section 9.1.1 [CSS2]) including the size of a rendered scroll bar (if any). For paged media, this is the width of the page box (as described by CSS2, section 13.2 [CSS2]).

<length>s are interpreted according to § 1.3 Units.

‘width’ is false in the negative range.

EXAMPLE 25

For example, this media query expresses that the style sheet is used on printed output wider than 25cm:

```
<link rel="stylesheet" media="print and (min-width: 25cm)" href="http://…" />
```

EXAMPLE 26

This media query expresses that the style sheet is used on devices with viewport (the part of the screen/paper where the document is rendered) widths between 400 and 700 pixels:

```
@media (400px <= width <= 700px) { … }
```

EXAMPLE 27

This media query expresses that style sheet is used if the width of the viewport is greater than 20em.

```
@media (min-width: 20em) { … }
```

The ‘em’ value is relative to the initial value of ‘font-size’.

## § 4.2. Height: the ‘height’ feature

| | |
|---|---|
| *Name:* | ***‘height’*** |
| *For:* | ‘@media’ |
| *Value:* | <length> |
| *Type:* | range |

The ‘height’ media feature describes the height of the targeted display area of the output device. For continuous media, this is the height of the viewport including the size of a rendered scroll bar (if any). For paged media, this is the height of the page box.

<length>s are interpreted according to § 1.3 Units.

‘height’ is false in the negative range.

## § 4.3. Aspect-Ratio: the ‘aspect-ratio’ feature

| | |
|---|---|
| *Name:* | ***‘aspect-ratio’*** |
| *For:* | ‘@media’ |
| *Value:* | <ratio> |
| *Type:* | range |

The ‘aspect-ratio’ media feature is defined as the ratio of the value of the ‘width’ media feature to the value of the ‘height’ media feature.

## § 4.4. Orientation: the ‘orientation’ feature

| | |
|---|---|
| *Name:* | ***‘orientation’*** |
| *For:* | ‘@media’ |
| *Value:* | portrait \| landscape |
| *Type:* | discrete |

***‘portrait’***

The ‘orientation’ media feature is ‘portrait’ when the value of the ‘height’ media feature is greater than or equal to the value of the ‘width’ media feature.

***‘landscape’***

Otherwise ‘orientation’ is ‘landscape’.

EXAMPLE 28

The following media query tests for “portrait” orientation, like a phone held upright.

```
@media (orientation:portrait) { … }
```

## § 5. Display Quality Media Features

### § 5.1. Display Resolution: the 'resolution' feature

| | |
|---|---|
| *Name:* | ***'resolution'*** |
| *For:* | '@media' |
| *Value:* | <resolution> \| infinite |
| *Type:* | range |

The 'resolution' media feature describes the resolution of the output device, i.e. the density of the pixels, taking into account the page zoom but assuming a pinch zoom of 1.0.

The 'resolution' media feature is false in the negative range

When querying media with non-square pixels, 'resolution' queries the density in the vertical dimension.

For printers, this corresponds to the screening resolution (the resolution for printing dots of arbitrary color). Printers might have a different resolution for grayscale printing.

For output mediums that have no physical constraints on resolution (such as outputting to vector graphics), this feature must match the ***'infinite'*** value. For the purpose of evaluating this media feature in the range context, 'infinite' must be treated as larger than any possible <resolution>. (That is, a query like '(resolution > 1000dpi)' will be true for an 'infinite' media.)

EXAMPLE 29
This media query simply detects "high-resolution" screens (those with a hardware pixel to CSS 'px' ratio of at least 2):

```
@media (resolution >= 2dppx)
```

EXAMPLE 30

For example, this media query expresses that a style sheet is used on devices with resolution greater than 300 dots per CSS ‘in’:

```
@media print and (min-resolution: 300dpi) { … }
```

This media query is equivalent, but uses the CSS ‘cm’ unit:

```
@media print and (min-resolution: 118dpcm) { … }
```

<resolution> does not refer to the number of device pixels per physical length unit, but the number of device pixels per css unit. This mapping is done by the user agent, so it is always known to the user agent.

If the user agent either has no knowledge of the geometry of physical pixels, or knows about the geometry physical pixels and they are (close enough to) square, it would not map a different number of device pixels per css pixels along each axis, and the would therefore be no difference between the vertical and horizontal resolution.

Otherwise, if the UA chooses to map a different number along each axis, this would be to respond to physical pixels not being square either. How the UA comes to this knowledge is out of scope, but having enough information to take this decision, it can invert the mapping should the device be rotated 90 degrees.

## § 5.2. Display Type: the ‘scan’ feature

| | |
|---|---|
| *Name:* | ***‘scan’*** |
| *For:* | ‘@media’ |
| *Value:* | interlace \| progressive |
| *Type:* | discrete |

The ‘scan’ media feature describes the scanning process of some output devices.

***‘interlace’***

CRT and some types of plasma TV screens used “interlaced” rendering, where video frames alternated between specifying only the “even” lines on the screen and only the “odd” lines, exploiting various automatic mental image-correction abilities to produce smooth motion. This allowed them to simulate a higher FPS broadcast at half the bandwidth cost.

When displaying on interlaced screens, authors should avoid very fast movement across the screen to avoid "combing", and should ensure that details on the screen are wider than '1px' to avoid "twitter".

***'progressive'***

A screen using "progressive" rendering displays each screen fully, and needs no special treatment.

Most modern screens, and all computer screens, use progressive rendering.

EXAMPLE 31

For example, the "feet" of letters in serif fonts are very small features that can provoke "twitter" on interlaced devices. The 'scan' media feature can be used to detect this, and use an alternative font with less chance of "twitter":

```
@media (scan: interlace) { body { font-family: sans-serif; } }
```

Note: At the time of writing, all known implementations match `scan: progressive` rather than `scan: interlace`.

## § 5.3. Detecting Console Displays: the 'grid' feature

| | |
|---|---|
| *Name:* | ***'grid'*** |
| *For:* | '@media' |
| *Value:* | <mq-boolean> |
| *Type:* | discrete |

The 'grid' media feature is used to query whether the output device is grid or bitmap. If the output device is grid-based (e.g., a "tty" terminal, or a phone display with only one fixed font), the value will be 1. Otherwise, the value will be 0.

The ***'<mq-boolean>'*** value type is an <integer> with the value '0' or '1'. Any other integer value is invalid. Note that '-0' is always equivalent to '0' in CSS, and so is also accepted as a valid <mq-boolean> value.

Note: The <mq-boolean> type exists only for legacy purposes. If this feature were being designed today, it would instead use proper named keywords for its values.

EXAMPLE 32
Here is an example that detects a narrow console screen:

```
@media (grid) and (max-width: 15em) { … }
```

Note: At the time of writing, all known implementations match `grid: 0` rather than `grid: 1`.

## § 5.4. Display Update Frequency: the ‘update’ feature

| | |
|---|---|
| *Name:* | ***‘update’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| slow \| fast |
| *Type:* | discrete |

The ‘update’ media feature is used to query the ability of the output device to modify the appearance of content once it has been rendered. It accepts the following values:

***‘none’***
Once it has been rendered, the layout can no longer be updated. Example: documents printed on paper.

***‘slow’***
The layout may change dynamically according to the usual rules of CSS, but the output device is not able to render or display changes quickly enough for them to be perceived as a smooth animation. Example: E-ink screens or severely under-powered devices.

***‘fast’***
The layout may change dynamically according to the usual rules of CSS, and the output device is not unusually constrained in speed, so regularly-updating things like CSS Animations can be used. Example: computer screens.

EXAMPLE 33

For example, if a page styles its links to only add underlines on hover, it may want to always display underlines when printed:

```
@media (update) {
  a { text-decoration: none; }
  a:hover, a:focus { text-decoration: underline; }
}
/* In non-updating UAs, the links get their default underline at all times. */
```

## § 5.5. Block-Axis Overflow: the ‘overflow-block’ feature

| | |
|---|---|
| *Name:* | ***‘overflow-block’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| scroll \| paged |
| *Type:* | discrete |

The ‘overflow-block’ media feature describes the behavior of the device when content overflows the initial containing block in the block axis.

***‘none’***
: There is no affordance for overflow in the block axis; any overflowing content is simply not displayed. Examples: billboards

***‘scroll’***
: Overflowing content in the block axis is exposed by allowing users to scroll to it. Examples: computer screens

***‘paged’***
: Content is broken up into discrete pages; content that overflows one page in the block axis is displayed on the following page. Examples: printers, ebook readers

Media that match ‘none’ or ‘scroll’ are said to be ***continuous media***, while those that match ‘paged’ are said to be ***paged media***

Note: Additional values for this media feature may be added in the future to describe classes of user agents with a hybrid behavior combining aspects of continuous and paged media. For example, the Presto layout engine (now discontinued) shipped with a semi-paginated presentation-mode behavior similar to ‘continuous’ except that it honored forced page breaks. Not knowing of any currently-shipping user agent with this type of behavior, the Working Group has decided not to add such a value in this level to avoid miscaracterizing any such user agent. Anyone implementing a user agent not adequately described by any of the values specified above is encouraged to contact the Working Group so that extensions to this media feature may be considered.

### § 5.6. Inline-Axis Overflow: the ‘overflow-inline’ feature

| | |
|---|---|
| *Name:* | ***‘overflow-inline’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| scroll |
| *Type:* | discrete |

The ‘overflow-inline’ media feature describes the behavior of the device when content overflows the initial containing block in the inline axis.

***‘none’***
: There is no affordance for overflow in the inline axis; any overflowing content is simply not displayed.

***‘scroll’***
: Overflowing content in the inline axis is exposed by allowing users to scroll to it.

Note: There are no known implementations of paged overflow of inline-overflowing content, and the very concept doesn’t seem to make much sense, so there is intentionally no ‘paged’ value for ‘overflow-inline’.

## § 6. Color Media Features

### § 6.1. Color Depth: the ‘color’ feature

| | |
|---|---|
| *Name:* | ***'color'*** |
| *For:* | '@media' |
| *Value:* | <integer> |
| *Type:* | range |

The 'color' media feature describes the number of bits per color component of the output device. If the device is not a color device, the value is zero.

'color' is false in the negative range.

EXAMPLE 34

For example, these two media queries express that a style sheet applies to all color devices:

```
@media (color) { … }
@media (min-color: 1) { … }
```

EXAMPLE 35

This media query expresses that a style sheet applies to color devices with at least 8 bits per color component:

```
@media (color >= 8) { … }
```

If different color components are represented by different number of bits, the smallest number is used.

EXAMPLE 36

For instance, if an 8-bit color system represents the red component with 3 bits, the green component with 3 bits, and the blue component with 2 bits, the 'color' media feature will have a value of 2.

In a device with indexed colors, the minimum number of bits per color component in the lookup table is used.

Note: The described functionality is only able to describe color capabilities at a superficial level. ‘color-gamut’, is generally more relevant to authors’ needs. If further functionality is required, RFC2879 [RFC2879] provides more specific media features which may be supported at a later stage.

## § 6.2. Paletted Color Screens: the ‘color-index’ feature

| | |
|---|---|
| *Name:* | ***‘color-index’*** |
| *For:* | ‘@media’ |
| *Value:* | <integer> |
| *Type:* | range |

The ‘color-index’ media feature describes the number of entries in the color lookup table of the output device. If the device does not use a color lookup table, the value is zero.

‘color-index’ is false in the negative range.

EXAMPLE 37

For example, here are two ways to express that a style sheet applies to all color index devices:

```
@media (color-index) { … }
@media (color-index >= 1) { … }
```

EXAMPLE 38

This media query expresses that a style sheet applies to a color index device with 256 or more entries:

```
<?xml-stylesheet media="(min-color-index: 256)"
  href="http://www.example.com/…" ?>
```

## § 6.3. Monochrome Screens: the ‘monochrome’ feature

| | |
|---|---|
| *Name:* | ***'monochrome'*** |
| *For:* | '@media' |
| *Value:* | <integer> |
| *Type:* | range |

The 'monochrome' media feature describes the number of bits per pixel in a monochrome frame buffer. If the device is not a monochrome device, the output device value will be 0.

'monochrome' is false in the negative range.

EXAMPLE 39

For example, this is how to express that a style sheet applies to all monochrome devices:

```
@media (monochrome) { … }
```

EXAMPLE 40

Express that a style sheet applies to monochrome devices with more than 2 bits per pixels:

```
@media (monochrome >= 2) { … }
```

EXAMPLE 41

Express that there is one style sheet for color pages and another for monochrome:

```
<link rel="stylesheet" media="print and (color)" href="http://…" />
<link rel="stylesheet" media="print and (monochrome)" href="http://…" />
```

## § 6.4. Color Display Quality: the 'color-gamut' feature

| | |
|---|---|
| *Name:* | ***'color-gamut'*** |
| *For:* | '@media' |
| *Value:* | srgb \| p3 \| rec2020 |

| *Type:* | discrete |
|---|---|

The ‘color-gamut’ media feature describes the approximate range of colors that are supported by the UA and output device. That is, if the UA receives content with colors in the specified space it can cause the output device to render the appropriate color, or something appropriately close enough.

Note: The query uses approximate ranges for a few reasons. Firstly, there are a lot of differences in display hardware. For example, a device might claim to support "Rec. 2020", but actually renders a significantly lower range of the full gamut. Secondly, there are a lot of different color ranges that different devices support, and enumerating them all would be tedious. In most cases the author does not need to know the exact capabilities of the display, just whether it is better than sRGB, or significantly better than sRGB. That way they can serve appropriate images, tagged with color profiles, to the user.

***‘srgb’***

The UA and output device can support approximately the sRGB gamut or more.

Note: It is expected that the vast majority of color displays will be able to return true to a query of this type.

***‘p3’***

The UA and output device can support approximately the gamut specified by the DCI P3 Color Space or more.

Note: The ‘p3’ gamut is larger than and includes the ‘srgb’ gamut.

***‘rec2020’***

The UA and output device can support approximately the gamut specified by the ITU-R Recommendation BT.2020 Color Space or more.

Note: The ‘rec2020’ gamut is larger than and includes the ‘p3’ gamut.

The following table lists the primary colors of these color spaces in terms of their color space chromaticity coordinates, as defined in [COLORIMETRY].

| **Color Space** | **White Point** | | **Primaries** | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | **Red** | | **Green** | | **Blue** | |
| | $x_W$ | $y_W$ | $x_R$ | $y_R$ | $x_G$ | $y_G$ | $x_B$ | $y_B$ |

| Color Space | White Point | | Primaries | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Red | | Green | | Blue | |
| | $x_W$ | $y_W$ | $x_R$ | $y_R$ | $x_G$ | $y_G$ | $x_B$ | $y_B$ |
| srgb | 0.3127 | 0.3290 | 0.640 | 0.330 | 0.300 | 0.600 | 0.150 | 0.060 |
| p3 | 0.3127 | 0.3290 | 0.680 | 0.320 | 0.265 | 0.690 | 0.150 | 0.060 |
| rec2020 | 0.3127 | 0.3290 | 0.708 | 0.292 | 0.170 | 0.797 | 0.131 | 0.046 |

Note: The table above does not contains enough information to fully describe the color spaces, but is sufficient to determine whether an output device approximately covers their respective gamuts. See [SRGB] for more information on sRGB, [SMPTE-EG-432-1-2010] and [SMPTE-RP-431-2-2011] for more information on DCI P3, and [ITU-R-BT-2020-2] for more information on ITU-R Recommendation BT.2020.

EXAMPLE 42
For example, this media query applies when the display supports colors in the range of DCI P3:

```
@media (color-gamut: p3) { … }
```

Note: An output device can return true for multiple values of this media feature, if its full output gamut is large enough, or one gamut is a subset of another supported gamut. As a result, this feature is best used in an "ascending" fashion—set a base value when ‘(color-gamut: srgb)’ is true, then override it if ‘(color-gamut: p3)’ is true, etc.

Note: Some output devices, such as monochrome displays, cannot support even the ‘srgb’ gamut. To test for these devices, you can use this feature in a negated boolean-context fashion: ‘not (color-gamut)’.

## § 7. Interaction Media Features

The “interaction” media features reflect various aspects of how the user interacts with the page.

Typical examples of devices matching combinations of ‘pointer’ and ‘hover’:

| | ‘pointer: none’ | ‘pointer: coarse’ | ‘pointer: fine’ |
|---|---|---|---|
| ‘hover: none’ | keyboard-only controls, sequential/spatial (d-pad) focus navigation | smartphones, touch screens | basic stylus digitizers (Cintiq, Wacom, etc) |
| ‘hover: hover’ | | Nintendo Wii controller, Kinect | mouse, touch pad, advanced stylus digitizers (Surface, Samsung Note, Wacom Intuos Pro, etc) |

The ‘pointer’ and ‘hover’ features relate to the characteristics of the “primary” pointing device, while ‘any-pointer’ and ‘any-hover’ can be used to query the properties of all potentially available pointing devices.

Note: While this specification does not define how user agents should decide what the “primary” pointing device is, the expectation is that user agents should make this determination by combining knowledge about the device/environment they are running on, the number and type of pointing devices available, and a notion of which of these is generally and/or currently being used. In situations where the primary input mechanism for a device is not a pointing device, but there is a secondary – and less frequently used – input that is a pointing devices, the user agent may decide to treat the non-pointing device as the primary (resulting in 'pointer: none'). user agents may also decide to dynamically change what type of pointing device is deemed to be primary, in response to changes in the user environment or in the way the user is interacting with the UA.

Note: The ‘pointer’, ‘hover’, ‘any-pointer’ and ‘any-hover’ features only relate to the characteristics, or the complete absence, of pointing devices, and can not be used to detect the presence of non-pointing device input mechanisms such as keyboards. Authors should take into account the potential presence of non-pointing device inputs, regardless of which values are matched when querying these features.

While ‘pointer’ and ‘hover’ can be used to design the main style and interaction mode of the page to suit the primary input mechanism (based on the characteristics, or complete absence, of the primary pointing device), authors should strongly consider using ‘any-pointer’ and ‘any-hover’ to take into account all possible types of pointing devices that have been detected.

## § 7.1. Pointing Device Quality: the ‘pointer’ feature

| | |
|---|---|
| *Name:* | ***‘pointer’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| coarse \| fine |
| *Type:* | discrete |

The ‘pointer’ media feature is used to query the presence and accuracy of a pointing device such as a mouse. If multiple pointing devices are present, the ‘pointer’ media feature must reflect the characteristics of the “primary” pointing device, as determined by the user agent. (To query the capabilities of *any* available pointing devices, see the ‘any-pointer’ media feature.)

***‘none’***
: The primary input mechanism of the device does not include a pointing device.

***‘coarse’***
: The primary input mechanism of the device includes a pointing device of limited accuracy. Examples include touchscreens and motion-detection sensors (like the Kinect peripheral for the Xbox.)

***‘fine’***
: The primary input mechanism of the device includes an accurate pointing device. Examples include mice, touchpads, and drawing styluses.

Both ‘coarse’ and ‘fine’ indicate the presence of a pointing device, but differ in accuracy. A pointing device with which it would be difficult or impossible to reliably pick one of several small adjacent targets at a zoom factor of 1 would qualify as ‘coarse’. Changing the zoom level does not affect the value of this media feature.

Note: As the UA may provide the user with the ability to zoom, or as secondary pointing devices may have a different accuracy, the user may be able to perform accurate clicks even if the value of this media feature is 'coarse'. This media feature does not indicate that the user will never be able to click accurately, only that it is inconvenient for them to do so. Authors are expected to react to a value of 'coarse' by designing pages that do not rely on accurate clicking to be operated.

For accessibility reasons, even on devices whose pointing device can be described as 'fine', the UA may give a value of 'coarse' or 'none' to this media query, to indicate that the user has difficulties manipulating the pointing device accurately or at all. In addition, even if the primary pointing device has 'fine' pointing accuracy, there may be additional 'coarse' pointing devices available to the user. Authors may wish to query the 'any-pointer' media feature to take these other 'coarse' potential pointing devices into account.

EXAMPLE 43

```
/* Make radio buttons and check boxes larger if we have an inaccurate primary po
@media (pointer:coarse) {
  input[type="checkbox"], input[type="radio"] {
    min-width:30px;
    min-height:40px;
    background:transparent;
  }
}
```

## § 7.2. Hover Capability: the 'hover' feature

| | |
|---|---|
| *Name:* | ***'hover'*** |
| *For:* | '@media' |
| *Value:* | none \| hover |
| *Type:* | discrete |

The 'hover' media feature is used to query the user's ability to hover over elements on the page with the primary pointing device. If a device has multiple pointing devices, the 'hover' media feature must reflect the characteristics of the "primary" pointing device, as determined by the user agent. (To query the capabilities of *any* available pointing devices, see the 'any-hover' media feature.)

***'none'***

Indicates that the primary pointing device can't hover, or that there is no pointing device. Examples include touchscreens and screens that use a basic drawing stylus.

Pointing devices that can hover, but for which doing so is inconvenient and not part of the normal way they are used, also match this value. For example, a touchscreen where a long press is treated as hovering would match 'hover: none'.

***'hover'***

Indicates that the primary pointing device can easily hover over parts of the page. Examples include mice and devices that physically point at the screen, like the Nintendo Wii controller.

EXAMPLE 44

For example, on a touch screen device that can also be controlled by an optional mouse, the 'hover' media feature should match 'hover: none', as the primary pointing device (the touch screen) does not allow the user to hover.

However, despite this, the optional mouse does allow users to hover. Authors should therefore be careful not to assume that the ':hover' pseudo class will never match on a device where 'hover:none' is true, but they should design layouts that do not depend on hovering to be fully usable.

For accessibility reasons, even on devices that do support hovering, the UA may give a value of 'hover: none' to this media query, to opt into layouts that work well without hovering. Note that even if the primary input mechanism has 'hover: hover' capability, there may be additional input mechanisms available to the user that do not provide hover capabilities.

EXAMPLE 45

```
/* Only use a hover-activated drop down menu on devices that can conveniently ho
@media (hover) {
  .menu > li        {display:inline-block;}
  .menu ul          {display:none; position:absolute;}
  .menu li:hover ul {display:block; list-style:none; padding:0;}
  /* ... */
}
```

## § 7.3. All Available Interaction Capabilities: the 'any-pointer' and 'any-hover' features

| | |
|---|---|
| *Name:* | ***‘any-pointer’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| coarse \| fine |
| *Type:* | discrete |

| | |
|---|---|
| *Name:* | ***‘any-hover’*** |
| *For:* | ‘@media’ |
| *Value:* | none \| hover |
| *Type:* | discrete |

The ‘any-pointer’ and ‘any-hover’ media features are identical to the ‘pointer’ and ‘hover’ media features, but they correspond to the union of capabilities of all the pointing devices available to the user. In the case of ‘any-pointer’, more than one of the values can match, if different pointing devices have different characteristics.

‘any-pointer’ and ‘any-hover’ must only match ‘none’ if *all* of the pointing devices would match ‘none’ for the corresponding query, or there are no pointing devices at all.

‘any-pointer’ is used to query the presence and accuracy of pointing devices. It does not take into account any additional non-pointing device inputs, and can not be used to test for the presence of other input mechanisms, such as d-pads or keyboard-only controls, that don’t move an on-screen pointer. 'any-pointer:none' will only evaluate to true if there are no pointing devices at all present.

EXAMPLE 46

On a traditional desktop environment with a mouse and keyboard, 'any-pointer:none' will be false (due to the presence of the mouse), even though a non-pointer input (the keyboard) is also present.

'any-hover:none' will only evaluate to true if there are no pointing devices, or if all the pointing devices present lack hover capabilities. As such, it should be understood as a query to test if any hover-capable pointing devices are present, rather than whether or not any of the pointing devices is hover-incapable. The latter scenario can currently not be determined using ‘any-hover’ or any other interaction media feature. Additionally, it does not take into account any non-pointing device inputs, such as d-pads or keyboard-only controls, which by their very nature are also not hover-capable.

EXAMPLE 47

On a touch-enabled laptop with a mouse and a touchscreen, 'any-hover:none' will evaluate to false (due to the presence of the hover-capable mouse), even though a non-hover-capable pointing device (the touchscreen) is also present. It is currently not possible to provide different styles for cases where different pointing devices have different hover capabilities.

Designing a page that relies on hovering or accurate pointing only because ‘any-hover’ or ‘any-pointer’ indicate that at least one of the available input mechanisms has these capabilities is likely to result in a poor experience. However, authors may use this information to inform their decision about the style and functionality they wish to provide based on any additional pointing devices that are available to the user.

EXAMPLE 48

A number of smart TVs come with a way to control an on-screen cursor, but it is often fairly basic controller which is difficult to operate accurately.

A browser in such a smart TV would have ‘coarse’ as the value of both ‘pointer’ and ‘any-pointer’, allowing authors to provide a layout with large and easy to reach click targets.

The user may also have paired a Bluetooth mouse with the TV, and occasionally use it for extra convenience, but this mouse is not the main way the TV is operated. ‘pointer’ still matches ‘coarse’, while ‘any-pointer’ now both matches ‘coarse’ and ‘fine’.

Switching to small click targets based on the fact that ‘(any-pointer: fine)’ is now true would not be appropriate. It would not only surprise the user by providing an experience out of line with what they expect on a TV, but may also be quite inconvenient: the mouse, not being the primary way to control the TV, may be out of reach, hidden under one of the cushions on the sofa...

By contrast, consider scrolling on the same TV. Scrollbars are difficult to manipulate without an accurate pointing device. Having prepared an alternative way to indicate that there is more content to be seen based on ‘(pointer: coarse)’ being true, an author may want to still show the scrollbars in addition if ‘(any-pointer: fine)’ is true, or to hide them altogether to reduce visual clutter if ‘(any-pointer: fine)’ is false.

## § Appendix A: Deprecated Media Features

The following media features are **deprecated**. They are kept for backward compatibility, but are not appropriate for newly written style sheets. Authors must not use them. User agents must support them as specified.

To query for the size of the viewport (or the page box on page media), the ‘width’, ‘height’ and ‘aspect-ratio’ media features should be used, rather than ‘device-width’, ‘device-height’ and ‘device-aspect-ratio’, which refer to the physical size of the the device regardless of how much space is available for the document being laid out. The device-* media features are also sometimes used as a proxy to detect mobile devices. Instead, authors should use media features that better represent the aspect of the device that they are attempting to style against.

### § device-width

| | |
|---|---|
| *Name:* | ***'device-width'*** |
| *For:* | '@media' |
| *Value:* | <length> |
| *Type:* | range |

The 'device-width' media feature describes the width of the rendering surface of the output device. For continuous media, this is the width of the Web-exposed screen area. For paged media, this is the width of the page sheet size.

'device-width' is false in the negative range.

EXAMPLE 49

```
@media (device-width < 800px) { … }
```

In the example above, the style sheet will apply only to screens less than '800px' in length. The 'px' unit is of the logical kind, as described in the Units section.

Note: If a device can be used in multiple orientations, such as portrait and landscape, the 'device-*' media features reflect the current orientation.

### § device-height

| | |
|---|---|
| *Name:* | ***'device-height'*** |
| *For:* | '@media' |
| *Value:* | <length> |
| *Type:* | range |

The 'device-height' media feature describes the height of the rendering surface of the output device. For continuous media, this is the height of the Web-exposed screen area. For paged media, this is the height of the page sheet size.

‘device-height’ is false in the negative range.

EXAMPLE 50

```
<link rel="stylesheet" media="(device-height > 600px)" />
```

In the example above, the style sheet will apply only to screens taller than 600 vertical pixels. Note that the definition of the ‘px’ unit is the same as in other parts of CSS.

### § device-aspect-ratio

| | |
|---|---|
| *Name:* | ***‘device-aspect-ratio’*** |
| *For:* | ‘@media’ |
| *Value:* | <ratio> |
| *Type:* | range |

The 'device-aspect-ratio media feature is defined as the ratio of the value of the ‘device-width’ media feature to the value of the 'device-height media feature.

EXAMPLE 51

For example, if a screen device with square pixels has 1280 horizontal pixels and 720 vertical pixels (commonly referred to as “16:9”), the following media queries will all match the device:

```
@media (device-aspect-ratio: 16/9) { … }
@media (device-aspect-ratio: 32/18) { … }
@media (device-aspect-ratio: 1280/720) { … }
@media (device-aspect-ratio: 2560/1440) { … }
```

## § Changes

### § Changes since the 5 September 2017 Candidate Recommendation

The following changes were made to this specification since the 5 September 2017 Candidate Recommendation:

- Deprecate the ‘speech’ media type. As media types are exclusive, it cannot be about screen readers, which as their name indicates, work based on a screen rendition, and therefore match the ‘screen’ media type. It could have been about pure-audio UAs, but no such implementation is known.
- Add note referencing the syntax spec to remind that token parsing is ascii case insensitive
- Fix a bug in the grammar that accidentally allowed forms like (width 500px), without any comparison
- Delegate the definition of <ratio> to [CSS-VALUES-4], as it is now used by more than just mediaqueries.

  > Note: [CSS-VALUES-4] has expanded the definition from `<ratio> = <integer> / <integer>` to `<ratio> = <number [0,∞]> [ / <number [0,∞]> ]?`

- Various editorial tweaks, phrasing improvements, and clarifications.
- Add definitions for the terms continuous media and paged media.
- Dropped the `optional-paged` value of ‘overflow-block’ due to a lack of current UAs having the behavior that it described.
- Mark ‘update’ at risk.

## § Changes since the 19 May 2017 Working Draft

The following changes were made to this specification since the 19 May 2017 Working Draft :

- Changed range media features to be false in the negative range instead of failing to parse negative values.
- Included enough information about the color spaces needed by ‘color-gamut’ directly into the specification.
- Marked ‘hover’, ‘pointer’, ‘any-hover’, and ‘any-pointer’ as no longer at-risk.

## § Changes Since Media Queries Level 3

The following changes were made to this specification since the 19 June 2012 Recommendation of Media Queries Level 3:

- Large editorial rewrite and reorganization of the document.

- Boolean-context media features are now additionally false if they would be true for the keyword none.
- Media features with numeric values can now be written in a range context.
- The ‘pointer’, ‘any-pointer’, ‘hover’, ‘any-hover’, ‘update’, ‘color-gamut’, ‘overflow-block’, and ‘overflow-inline’ media features were added.
- ‘or’, ‘and’, ‘only’ and ‘not’ are disallowed from being recognized as media types, even invalid ones. (They’ll trigger a syntax error instead.)
- All media types except for ‘screen’, ‘print’, ‘speech’, and ‘all’ are deprecated.
- Deprecated ‘device-width’, ‘device-height’, ‘device-aspect-ratio’, and made them refer to the Web-exposed screen area instead of the screen for privacy and security reasons.
- Mediaqueries may depend on the evaluation of style sheets in some cases

## § Acknowledgments

This specification is the product of the W3C Working Group on Cascading Style Sheets.

Comments from Amelia Bellamy-Royds, Andreas Lind, Andres Galante, Arve Bersvendsen, Björn Höhrmann, Chris Lilley, Chris Rebert, Christian Biesinger, Christoph Päper, Dean Jackson, Elika J. Etemad (fantasai), Emilio Cobos Álvarez, François Remy, Frédéric Wang, Greg Whitworth, Ian Pouncey, James Craig, Jinfeng Ma, Kivi Shapiro, L. David Baron, Masataka Yakura, Melinda Grant, Michael[tm] Smith, Nicholas C. Zakas Patrick H. Lauke, Philipp Hoschka, Rick Byers, Rijk van Geijtenbeek, Roger Gimson, Sam Sneddon, Sigurd Lerstad, Simon Kissane, Simon Pieters, Steven Pemberton, Susan Lesch, Tantek Çelik, Thomas Wisniewski, Vi Nguyen, Xidorn Quan, Yves Lafon, and 張俊芝 improved this specification.

## § 8. Privacy and Security Considerations

This specification introduces no new security considerations.

Media Queries enable CSS to query various aspects of the page’s environment, including things that can be difficult or impossible to find via scripting. This is potentially a privacy hazard, allowing enhanced fingerprinting of a user, but the risk is generally low. At minimum, the same information should be *inferrable* via scripting by examining the User Agent string. However, UA string spoofing does not affect Media Queries, making this a somewhat more robust detection technique.

That said, the information granted by Media Queries is relatively coarse, and does not contribute much entropy in this regard.

A few legacy Media Features (‘device-width’, ‘device-height’, and ‘device-aspect-ratio’) expose information about the environment in which the UA is running without any clear benefit to doing so. They are retained for compatibility reasons, but for the sake of privacy and security, UAs have been allowed to report inaccurate information.

The TAG has developed a self-review questionnaire to help editors and Working Groups evaluate the risks introduced by their specifications. Answers are provided below.

**Does this specification deal with personally-identifiable information?**
No.

**Does this specification deal with high-value data?**
No.

**Does this specification introduce new state for an origin that persists across browsing sessions?**
No.

**Does this specification expose persistent, cross-origin state to the web?**
No.

**Does this specification expose any other data to an origin that it doesn’t currently have access to?**
No.

**Does this specification enable new script execution/loading mechanisms?**
No.

**Does this specification allow an origin access to a user’s location?**
No.

**Does this specification allow an origin access to sensors on a user’s device?**
No.

**Does this specification allow an origin access to aspects of a user’s local computing environment?**
Yes, as described in the prose above this questionnaire.

**Does this specification allow an origin access to other devices?**
No.

**Does this specification allow an origin some measure of control over a user agent’s native UI?**
No.

**Does this specification expose temporary identifiers to the web?**
No.

**Does this specification distinguish between behavior in first-party and third-party contexts?**
No.

**How should this specification work in the context of a user agent’s "incognito" mode?**

No difference in behavior is needed.

**Does this specification persist data to a user’s local device?**

No.

**Does this specification have a "Security Considerations" and "Privacy Considerations" section?**

Yes, this is the section you are currently reading.

**Does this specification allow downgrading default security characteristics?**

No.

# § 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 52

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 http://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

## § Terms defined by reference

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

- @import
- initial value

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

- @media

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

- font-size

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

- <any-value>
- <delim-token>
- <function-token>
- parse a comma-separated list of component values

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

- <dimension>
- <ident>
- <integer>
- <length>
- <number>
- <resolution>
- ?
- cm
- em
- in
- px
- relative length
- |

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

- <ratio>

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

- block axis
- inline axis

[cssom-view-1] defines the following terms:

- page zoom
- pinch zoom
- web-exposed screen area

[INFRA] defines the following terms:

- ascii case-insensitive

## § References

## § Normative References

**[COLORIMETRY]**
Colorimetry, Fourth Edition. CIE 015:2018. 2018. URL: http://www.cie.co.at/publications/colorimetry-4th-edition

**[CSS-CASCADE-5]**
Elika Etemad; Miriam Suzanne; Tab Atkins Jr.. CSS Cascading and Inheritance Level 5. 19 January 2021. WD. URL: https://www.w3.org/TR/css-cascade-5/

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

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

**[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-VALUES-4]**
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 4. 11 November 2020. WD. URL: https://www.w3.org/TR/css-values-4/

**[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/

**[CSS2]**
Bert Bos; et al. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. 7 June 2011. REC. URL: https://www.w3.org/TR/CSS21/

**[CSSOM-VIEW-1]**
Simon Pieters. CSSOM View Module. 17 March 2016. WD. URL: https://www.w3.org/TR/cssom-view-1/

**[MEDIAQUERIES-3]**
Florian Rivoal; et al. Media Queries. 19 June 2012. REC. URL: https://www.w3.org/TR/css3-mediaqueries/

**[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

## § Informative References

**[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/

**[HTML401]**
Dave Raggett; Arnaud Le Hors; Ian Jacobs. HTML 4.01 Specification. 27 March 2018. REC. URL: https://www.w3.org/TR/html401/

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

**[ITU-R-BT-2020-2]**
Parameter values for ultra-high definition television systems for production and international programme exchange. October 2015. URL: https://www.itu.int/rec/R-REC-BT.2020/en

**[RFC2879]**
G. Klyne; L. McIntyre. Content Feature Schema for Internet Fax (V2). August 2000. Proposed Standard. URL: https://tools.ietf.org/html/rfc2879

**[SMPTE-EG-432-1-2010]**
SMPTE Engineering Guideline - Digital Source Processing — Color Processing for D-Cinema. 2010. URL: http://ieeexplore.ieee.org/document/7289763/

**[SMPTE-RP-431-2-2011]**
SMPTE Recommended Practice - D-Cinema Quality — Reference Projector and Environment. 2011. URL: http://ieeexplore.ieee.org/document/7290729/

**[SRGB]**
Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. URL: https://webstore.iec.ch/publication/6169

**[XML-STYLESHEET]**
James Clark; Simon Pieters; Henry Thompson. Associating Style Sheets with XML documents 1.0 (Second Edition). 28 October 2010. REC. URL: https://www.w3.org/TR/xml-stylesheet/

## § Property Index

No properties defined.

## § ‘@media’ Descriptors

| Name | Value | Initial | Type |
|---|---|---|---|
| ‘any-hover’ | none \| hover | | discrete |

| Name | Value | Initial | Type |
|---|---|---|---|
| ‘any-pointer’ | none \| coarse \| fine | | discrete |
| ‘aspect-ratio’ | <ratio> | | range |
| ‘color’ | <integer> | | range |
| ‘color-gamut’ | srgb \| p3 \| rec2020 | | discrete |
| ‘color-index’ | <integer> | | range |
| ‘device-aspect-ratio’ | <ratio> | | range |
| ‘device-height’ | <length> | | range |
| ‘device-width’ | <length> | | range |
| ‘grid’ | <mq-boolean> | | discrete |
| ‘height’ | <length> | | range |
| ‘hover’ | none \| hover | | discrete |
| ‘monochrome’ | <integer> | | range |
| ‘orientation’ | portrait \| landscape | | discrete |
| ‘overflow-block’ | none \| scroll \| paged | | discrete |
| ‘overflow-inline’ | none \| scroll | | discrete |
| ‘pointer’ | none \| coarse \| fine | | discrete |
| ‘resolution’ | <resolution> \| infinite | | range |
| ‘scan’ | interlace \| progressive | | discrete |
| ‘update’ | none \| slow \| fast | | discrete |
| ‘width’ | <length> | | range |