



# Media Queries Level 4

## W3C First Public Working Draft, 3 June 2014

**This version:**

<http://www.w3.org/TR/2014/WD-mediaqueries-4-20140603/>

**Latest version:**

<http://www.w3.org/TR/mediaqueries4/>

**Editor's Draft:**

<http://dev.w3.org/csswg/mediaqueries4/>

**Feedback:**

[www-style@w3.org](mailto:www-style@w3.org) with subject line "[mediaqueries] ... *message topic* ..." ([archives](#))

**Editors:**

[Florian Rivoal](#) (Invited Expert)

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

**Former Editors:**

[Håkon Wium Lie](#) (Opera)

[Tantek Çelik](#) (Mozilla)

[Daniel Glazman](#) (Disruptive Innovations)

[Anne van Kesteren](#) (Mozilla)

**Issue Tracking:**

[Inline](#)

Copyright © 2014 W3C® (MIT, [ERCIM](#), [Keio](#), [Beihang](#)), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use rules](#) apply.

---

## 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, in speech, etc.

## Status of this document

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

This document is a **First Public Working Draft**.

Publication as a First Public Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The (archived) public mailing list [www-style@w3.org](mailto:www-style@w3.org) (see [instructions](#)) is preferred for discussion of this specification. When sending e-mail, please put the text “mediaqueries” in the subject, preferably like this: “[mediaqueries] ...*summary of comment*...”

This document was produced by the CSS Working Group (part of the Style Activity).

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

## Table of Contents

- 1 Introduction**
  - 1.1 Module interactions
  - 1.2 Values
  - 1.3 Units
  
- 2 Media Queries**
  - 2.1 Combining Media Queries
  - 2.2 Media Query Modifiers
    - 2.2.1 *Negating a Media Query: the 'not' keyword*
    - 2.2.2 *Hiding a Media Query From Legacy User Agents: the 'only' keyword*
  - 2.3 Media Types
  - 2.4 Media Features
    - 2.4.1 *Media Feature Types: "range" and "discrete"*
    - 2.4.2 *Evaluating Media Features in a Boolean Context*
    - 2.4.3 *Evaluating Media Features in a Range Context*
    - 2.4.4 *Using "min-" and "max-" Prefixes On Range Features*
  
- 3 Syntax**
  - 3.1 Error Handling
  
- 4 Screen/Device Dimensions Media Features**
  - 4.1 width
  - 4.2 height
  - 4.3 aspect-ratio
  - 4.4 orientation
  - 4.5 device-width
  - 4.6 device-height
  - 4.7 device-aspect-ratio
  
- 5 Display Quality Media Features**
  - 5.1 resolution
  - 5.2 scan
  - 5.3 grid
  - 5.4 update-frequency
  - 5.5 overflow-block
  - 5.6 overflow-inline
  
- 6 Color Media Features**
  - 6.1 color

- 6.2 color-index
- 6.3 monochrome
- 7 Interaction Media Features**
  - 7.1 pointer
  - 7.2 hover
  - 7.3 any-pointer and any-hover
- 8 Environment Media Features**
  - 8.1 light-level
- 9 Scripting Media Features**
  - 9.1 scripting
- 10 Assorted Issues**
- 11 Custom Media Queries**
  - 11.1 Script-based Custom Media Queries
  - 11.2 CSSOM

## **Changes**

Changes Since the Media Queries Level 3

## **Acknowledgments**

## **Conformance**

Document conventions

Conformance classes

Partial implementations

Experimental implementations

Non-experimental implementations

## **References**

Normative References

Informative References

## **Index**

## **Property index**

‘@media’ Descriptors

## Issues Index

### § 1 Introduction

*This section is not normative.*

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 [XMLSTYLE] 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) { ... }
```

Note: The [XMLSTYLE] specification has not yet been updated to use media queries in the media pseudo-attribute.

## § 1.1 Module interactions

This module replaces and extends the Media Queries, Media Type and Media Features defined in [CSS21] sections 7 and in [MEDIAQ].

## § 1.2 Values

Value types not defined in this specification, such as <integer>, <number> or <resolution>, are defined in [CSS3VAL]. Other CSS modules may expand the definitions of these value types.

This specification also introduces some new value types.

The **<ratio>** value type is a positive (not zero or negative) <integer> followed by optional whitespace, followed by a solidus ('/'), followed by optional whitespace, followed by a positive <integer>. <ratio>s can be ordered or compared by transforming them into the number obtained by dividing their first <integer> by their second <integer>.

**ISSUE 1** Reasonable to restrict `<ratio>` to `<integer>`s? I've seen aspect ratios written with decimal points in real life.

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.)

## § 1.3 Units

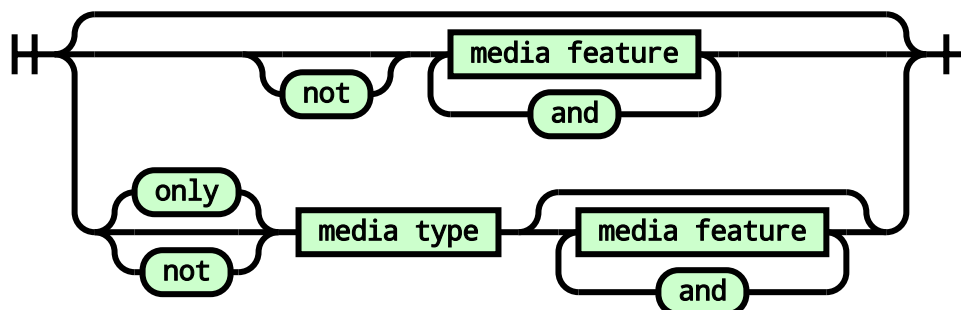
The units used in media queries are the same as in other parts of CSS, as defined in [CSS3VAL]. For example, the pixel unit represents CSS pixels and not physical pixels.

Relative 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, such as the `'@viewport'` rule.

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
- all specified media features are 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 the error handling section. 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 should re-evaluate media queries in response to changes in the user environment, 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.

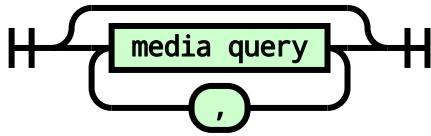
Unless another feature explicitly specifies that it affects the resolution of Media Queries, it is never necessary to apply a style sheet in order to evaluate expressions.

Note: CSS Device Adaptation [CSS-DEVICE-ADAPT]] defines how `'@viewport'` rules interact with Media Queries.

## § 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’ or ‘speech’.

### ***speech***

Matches screenreaders and similar devices that “read out” a page.

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.

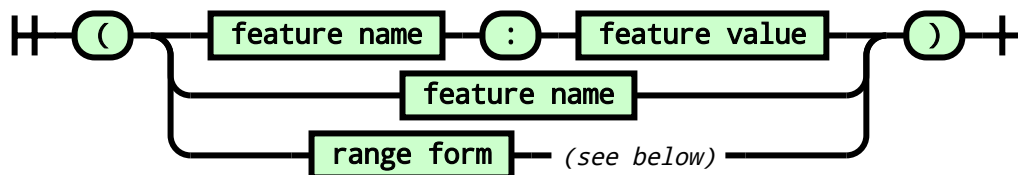
- *tty*
- *tv*
- *projection*
- *handheld*
- *braille*
- *embossed*
- *aural*

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' keyword, 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 is 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 does not apply to the device where the UA is running, that media feature will always be 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 'light-level' or 'scripting', 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 the number `'0'`, a dimension with the value `'0'`, or the keyword `'none'`, the media feature evaluates to false. If it would be true for any values **other than** the above, it evaluates to true. Otherwise, it evaluates to false.

Note: The “trichotomy” above allows for correct handling of MQs as false when on devices where they don't apply at all, such as `'scan'` on a speech device.

### EXAMPLE 11

Some media features are designed to be written like this.

For example, `'scripting'` is typically written as `'(scripting)'` to test if scripting is enabled, or `'not (scripting)'` to see if it's disabled.

It can still be given an explicit value as well, with `'(scripting: enabled)'` equal to `'(scripting)'`, and `'(scripting: none)'` equal to `'not (scripting)'`.

### 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.

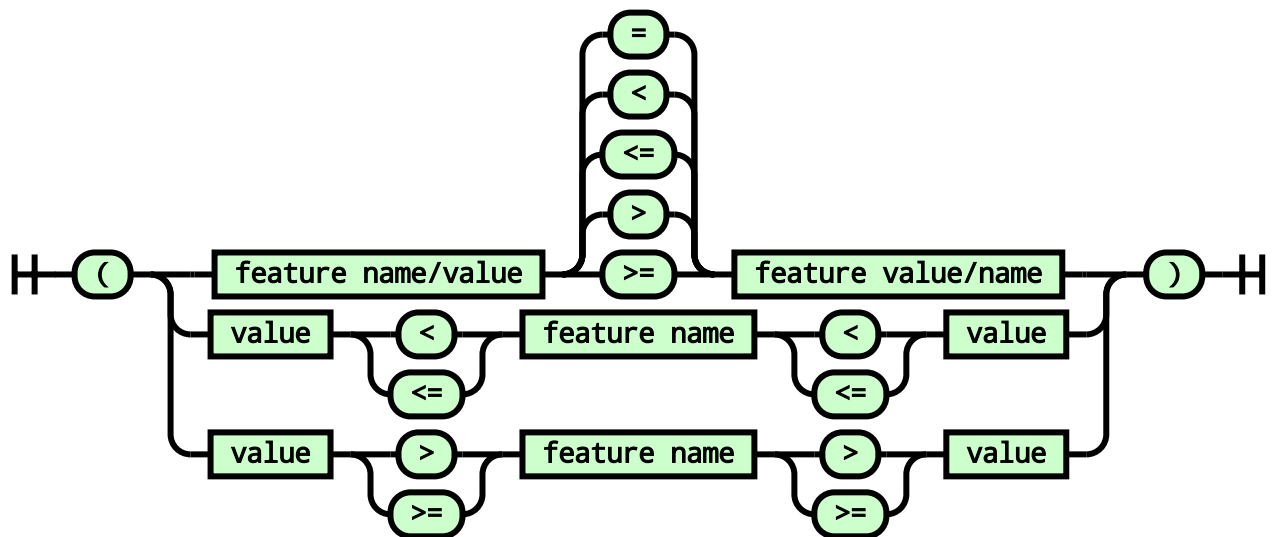
**EXAMPLE 13**

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:



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).

### § 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)'`.

“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 16

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'`.)

## § 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 [CSS3SYN] and [CSS3VAL].

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.

```

<media-query> = <media-condition>?
                | [ not | only ]? <media-type> [ and <media-condition> ]?
<media-type> = <ident>

<media-condition> = <media-not> | <media-and> | <media-or> | <media-in-parens>
<media-not> = not <media-in-parens>
<media-and> = <media-in-parens> [ and <media-in-parens> ]+
<media-or> = <media-in-parens> [ 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-value>
                | <mf-value> [ '<' | '>' ]? '='? <mf-name>
                | <mf-value> '<' '='? <mf-name> '<' '='? <mf-value>
                | <mf-value> '>' '='? <mf-name> '>' '='? <mf-value>
<mf-name> = <ident>
<mf-value> = <number> | <dimension> | <ident> | <ratio>

<general-enclosed> = [ <function-token> | ( ) <any-value>* )

```

The `<media-type>` production does not include the keywords `'only'`, `'not'`, `'and'`, and `'or'`. A `<dimension>` is a `dimension`. An `<ident>` is an `identifier`.

No whitespace is allowed between the `"<"` or `">"` `<delim-token>`s and the following `"="` `<delim-token>`, if it's present. Whitespace *must* be present between a `)` character and a `'not'`, `'and'`, or `'or'` keyword, and between a `'not'`, `'and'`, or `'or'` keyword and a `(` character.

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. )`

In addition to conforming to the syntax, each media query needs to use media types and media features according to their respective specification in order to be considered conforming.

**EXAMPLE 17**

Only the first media query is conforming in the example below because the "example" media type does not exist.

```
@media all { body { background:lime } }  
@media example { body { background:red } }
```

Each of the major terms of <media-condition> is associated with a boolean result, as follows:

**<media-condition>**

The result is the result of the child term.

**<media-not>**

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

**<media-and>**

The result is true if all of the <media-in-parens> child terms are true, and false otherwise.

**<media-or>**

The result is true if any of the <media-in-parens> child terms are true, and false otherwise.

**<media-in-parens>**

If it contains a <media-in-parens> or <media-feature> child term, the result is the result of that child term. If it contains a <general-enclosed> child term, the result is false.

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.

## § 3.1 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 18

```
@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 19

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 20

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>, must make the entire <media-query> be replaced by **'not all'**.

### EXAMPLE 21

```
<link media="screen and (max-weight: 3kg) and (color), (color)"
      rel="stylesheet" href="example.css" />
```

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 22

```
@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 23

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

### EXAMPLE 24

This media query is turned into `'not all'` because negative lengths are not allowed for the `'width'` media feature:

```
@media (min-width: -100px) { ... }
```

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 Screen/Device Dimensions Media Features

## § 4.1 width

<i>Name:</i>	<b><i>width</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;length&gt;</u>
<i>Type:</i>	range

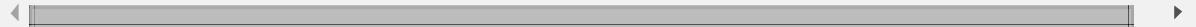
The 'width' media feature describes the width of the targeted display area of the output device. For continuous media, this is the width of the viewport (as described by CSS2, section 9.1.1 [CSS21]) including the size of a rendered scroll bar (if any). For paged media, this is the width of the page box (as described by CSS2, section 13.2 [CSS21]).

A specified <length> cannot be negative.

**EXAMPLE 25**

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

```
@media (min-width: 400px) and (max-width: 700px) { ... }
```

**EXAMPLE 27**

This media query expresses that style sheet is usable 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

<i>Name:</i>	<b><i>height</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;length&gt;</u>
<i>Type:</i>	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.

A specified <length> cannot be negative.

## § 4.3 aspect-ratio

<i>Name:</i>	<b><i>aspect-ratio</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;ratio&gt;</u>
<i>Type:</i>	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

<i>Name:</i>	<b><i>orientation</i></b>
--------------	---------------------------

<i>For:</i>	<u>'@media'</u>
-------------	-----------------

<i>Value:</i>	portrait   landscape
---------------	----------------------

<i>Type:</i>	discrete
--------------	----------

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. Otherwise 'orientation' is 'landscape'.

#### EXAMPLE 28

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

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

## § 4.5 device-width

<i>Name:</i>	<b><i>device-width</i></b>
--------------	----------------------------

<i>For:</i>	<u>'@media'</u>
-------------	-----------------

<i>Value:</i>	<u>&lt;length&gt;</u>
---------------	-----------------------

<i>Type:</i>	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 screen. For paged media, this is the width of the page sheet size.

A specified <length> cannot be negative.

**EXAMPLE 29**

```
@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.

**ISSUE 2** We should deprecate the device-\* media features. They don't do anything useful, and are mostly just abused as a ghetto "phone versus desktop" media type. We've now defined a number of useful MQs for differentiating devices in useful ways instead.

## § 4.6 device-height

<i>Name:</i>	<b><i>device-height</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;length&gt;</u>
<i>Type:</i>	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 screen. For paged media, this is the height of the page sheet size.

A specified <length> cannot be negative.

**EXAMPLE 30**

```
<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.



## § 4.7 device-aspect-ratio

<i>Name:</i>	<b><i>device-aspect-ratio</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;ratio&gt;</u>
<i>Type:</i>	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 31

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) { ... }
```

## § 5 Display Quality Media Features

### § 5.1 resolution

<i>Name:</i>	<b><i>resolution</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;resolution&gt;</u>
<i>Type:</i>	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.

When querying devices with non-square pixels, in 'min-resolution' queries the least-dense dimension must be compared to the specified value and in 'max-resolution' queries the most-dense dimensions must be compared instead. A 'resolution' query that's not evaluated in a range context never matches a device with non-square pixels.

**ISSUE 3** Figure out how to make the above work properly for `</>` syntax. Just translate it over directly? That prevents you from doing a "less than/greater than" dichotomy without using 'not'. Hmm.

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.

**ISSUE 4** Another media feature should probably be added to deal with the type of resolution authors want to know to deal with monochrome printing.

#### EXAMPLE 32

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 33

For example, this media query expresses that a style sheet is usable 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) { ... }
```

## § 5.2 scan

<i>Name:</i>	<b><i>scan</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	interlace   progressive
<i>Type:</i>	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 34**

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; } }
```

## § 5.3 grid

<i>Name:</i>	<b><i>grid</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;mq-boolean&gt;</u>
<i>Type:</i>	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.

#### EXAMPLE 35

Here is an example that detects a narrow console screen:

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

## § 5.4 update-frequency

<i>Name:</i>	<b><i>update-frequency</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   slow   normal
<i>Type:</i>	discrete

The 'update-frequency' 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.

### **normal**

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 36

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

```
a { text-decoration: none; }
a:hover, a:focus { text-decoration: underline; }
@media (update-frequency: none) {
  a { text-decoration: underline; }
}
```

## § 5.5 overflow-block

<i>Name:</i>	<b><i>overflow-block</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   scroll   optional-paged   paged
<i>Type:</i>	discrete

The 'overflow-block' media feature describes the behavior of the device when content overflows the viewport 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

### **optional-paged**

Overflowing content in the block axis is exposed by allowing users to scroll to it, but page breaks can be manually triggered (such as via 'break-inside'/etc) to cause the following content to display on the following page. Examples: slideshows

### ***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

**ISSUE 5** “Viewport” isn’t the right term here, or in 'overflow-inline'.

**ISSUE 6** The names of 'overflow-block' and 'overflow-inline' are possibly confusing. Any better names?

## § 5.6 overflow-inline

<i>Name:</i>	<b><i>overflow-inline</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   scroll
<i>Type:</i>	discrete

The 'overflow-inline' media feature describes the behavior of the device when content overflows the viewport 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

<i>Name:</i>	<b><i>color</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;integer&gt;</u>
<i>Type:</i>	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.

A specified <integer> cannot be negative.

#### EXAMPLE 37

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

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

#### EXAMPLE 38

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 39

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. If further functionality is required, RFC2531 [RFC2531] provides more specific media features which may be supported at a later stage.

## § 6.2 color-index

<i>Name:</i>	<b><i>color-index</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;integer&gt;</u>
<i>Type:</i>	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.

A specified <integer> cannot be negative.

### EXAMPLE 40

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 41

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



<i>Name:</i>	<b><i>monochrome</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	<u>&lt;integer&gt;</u>
<i>Type:</i>	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.

A specified <integer> cannot be negative.

#### EXAMPLE 42

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

```
@media (monochrome) { ... }
```

#### EXAMPLE 43

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

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

#### EXAMPLE 44

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://..." />
```

## § 7 Interaction Media Features

### § 7.1 pointer

<i>Name:</i>	<b><i>pointer</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   coarse   fine
<i>Type:</i>	discrete

The 'pointer' media feature is used to query about the presence and accuracy of a pointing device such as a mouse. If a device has multiple input mechanisms, the 'pointer' media feature must reflect the characteristics of the "primary" input mechanism, as determined by the user agent. (To query the capabilities of *any* available input mechanism, 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.

### ***fine***

The primary input mechanism of the device includes an accurate pointing device.

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.

Typical examples of devices matching combinations of 'pointer' and 'hover':

		<b>pointer</b>	
		<b>coarse</b>	<b>fine</b>
<b>hover</b>	<b>none</b>	smartphones, touch screens	stylus-based screens (Cintiq, Wacom, etc)
	<b>hover</b>	Nintendo Wii controller, Kinect	mouse, touch pad

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 input device accurately or at all.

#### EXAMPLE 45

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

## § 7.2 hover

<i>Name:</i>	<b><i>hover</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   on-demand   hover
<i>Type:</i>	discrete

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

***none***

Indicates that the primary pointing system can't hover, or there is no pointing system.

***on-demand***

Indicates that the primary pointing system can hover, but it requires a significant action on the user's part. For example, some devices can't normally hover, but will activate hover on a "long press".

***hover***

Indicates that the primary pointing system can easily hover over parts of the page.

**EXAMPLE 46**

For example, on a touch screen device that can also be controlled by an optional mouse, the 'hover' media feature should match 'none', as the primary interaction mode (touching the screen) can't hover.

Authors should therefore be careful not to assume that the `':hover'` pseudo class will never match on 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.

**EXAMPLE 47**

```
/* Only use a hover-activated drop down menu on devices that can hover. */
@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 any-pointer and any-hover

<i>Name:</i>	<b><i>any-pointer</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   coarse   fine
<i>Type:</i>	discrete

<i>Name:</i>	<b><i>any-hover</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   on-demand   hover
<i>Type:</i>	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 input devices available to the user. More than one of their values can match, if different input devices have different characteristics.

## § 8 Environment Media Features

### § 8.1 light-level

<i>Name:</i>	<b><i>light-level</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	dim   normal   washed
<i>Type:</i>	discrete

The 'light-level' media feature is used to query about the ambient light-level in which the device is used, to allow the author to adjust style of the document in response. The following values are valid:

***dim***

The device is used in a dim environment, where excessive contrast and brightness would be distracting or uncomfortable to the reader. For example: night time, or a dimly illuminated indoor environment.

***normal***

The device is used in a environment with a light level in the ideal range for the screen, and which does not necessitate any particular adjustment.

***washed***

The device is used in an exceptionally bright environment, causing the screen to be washed out and difficult to read. For example: bright daylight.

User agents should set the thresholds between the 3 levels in a way that takes into account the characteristics of the device.

Even though it is expected that User Agents will adjust the value of this media feature based on ambient light sensors, this specification intentionally refrains from defining the 3 levels in terms of a measurement in lux, for several reasons:

- Devices equipped with a light sensor usually adjust the brightness of the screen automatically. Depending on the level of adjustment, the thresholds for needing a low contrast or high contrast content may vary.
- Different screen technologies wash out at very different ambient light levels; e-ink displays remain readable in bright daylight, while liquid crystal displays do not.
- Many embedded light sensors are inaccurately calibrated, making it difficult to establish useful thresholds valid across devices.

For accessibility purposes, user agents may offer manual controls allowing the user to switch between the 3 levels of independently of the ambient light level, as high contrast or low contrast styles may be more suitable for users with visual disabilities.

**ISSUE 7** Using this media feature for accessibility purposes overlaps a lot with the high-contrast media feature proposed by Microsoft. Can we adjust this so that it covers all use cases for both, or somehow modify them to work in an orthogonal, rather than overlapping, fashion? Also, the high-contrast media feature could be extended to also cover inverted colors, as discussed in <http://lists.w3.org/Archives/Public/www-style/2013Oct/0672.html>

**EXAMPLE 48**

```
@media (light-level: normal) {  
  p { background: url("texture.jpg"); color: #333 }  
}  
@media (light-level: dim) {  
  p { background: #222; color: #ccc }  
}  
@media (light-level: washed) {  
  p { background: white; color: black; font-size: 2em; }  
}
```

## § 9 Scripting Media Features

### § 9.1 scripting

<i>Name:</i>	<b><i>scripting</i></b>
<i>For:</i>	<u>'@media'</u>
<i>Value:</i>	none   initial-only   enabled
<i>Type:</i>	discrete

The 'scripting' media feature is used to query whether scripting languages, such as JavaScript, are supported on the current document.

***enabled***

Indicates that the user agent supports scripting of the page and that support is active for the current document.

***initial-only***

Indicates that scripting is enabled during the initial page load, but is not supported afterwards. Examples are printed pages, or pre-rendering network proxies that render a page on a server and send a nearly-static version of the page to the user.

***none***

Indicates that the user agent will not run scripts for this document; either it doesn't

support a scripting language, or the support isn't active for the current document.

Some user agents have the ability to turn off scripting support on a per script basis or per domain basis, allowing some, but not all, scripts to run in a particular document. The 'scripting' media feature does not allow fine grained detection of which script is allowed to run. In this scenario, the value of the 'scripting' media feature should be 'enabled' if scripts originating on the same domain as the document are allowed to run, and 'none' otherwise.

Note: A future level of CSS may extend this media feature to allow fine-grained detection of which script is allowed to run.

**ISSUE 8** Is there a use-case for distinguishing between "UA doesn't support scripting" and "scripting is supported, but turned off"?

## § 10 Assorted Issues

**ISSUE 9** We need a media feature (or set of media features) to detect the type of keyboard available. It should be able to distinguish between full computer keyboards, phone dial pads, tv remotes, or virtual keyboards. As an attempt at an exhaustive list is likely to fail, finding atomic features to decompose these into would be preferable, but these remain to be identified.

- always vs in text forms only
- just numbers vs free alphanumeric input vs full ime support
- work properly vs horrible lag like on a tv remote

How much is actually useful for styling?



**ISSUE 10** Example sets of MQs that would match on different types of devices

- printer: update-frequency:none, pointer:none, hover:none, overflow-block: paged, overflow-inline: none
- eink with stylus: update-frequency:slow, pointer:fine, hover:none, overflow-block: paged, overflow-inline: none
- tv: update-frequency: normal, pointer: none, hover: none: overflow-block: none, overflow-inline: none
- tablet: update-frequency: normal, pointer coarse, hover none, overflow-block: scroll, overflow-inline: scroll
- wii: update-frequency normal, pointer: coarse, hover: yes, overflow-block: scroll, overflow-inline: scroll
- Chromebook pixel: update-frequency: normal, pointer: coarse \*and\* fine, hover: over, overflow-block: scroll, overflow-inline: scroll
- Glass: update-frequency: normal, pointer: coarse, hover: none, overflow-block: none, overflow-inline: none
- XTERM: update-frequency: normal, pointer:none, hover:none, grid:1, overflow-block: scroll, overflow-inline: none

**ISSUE 11** <http://lists.w3.org/Archives/Public/www-style/2013Mar/0448.html>

MQ for detecting if the device is willing to display/print backgrounds and other ink-hungry properties.

**ISSUE 12** What to do about the ‘view-mode’ media feature?

## § 11 Custom Media Queries

When designing documents that use media queries, the same media query may be used in multiple places, such as to qualify multiple ‘@import’ statements. Repeating the same media query multiple times is an editing hazard; an author making a change must edit every copy in the same way, or suffer from difficult-to-find bugs in their CSS.

To help ameliorate this, this specification defines a method of defining custom media queries, which are simply-named aliases for longer and more complex media queries. In this way, a media query used in multiple places can instead be assigned to a custom media query, which can be used everywhere, and editing the media query requires touching only one line of code.

A **custom media query** is defined with the '@custom-media' rule:

```
@custom-media = @custom-media <extension-name> <media-query-list> ;
```

This defines that the custom media query named by the <extension-name> represents the given <media-query-list>.

The <extension-name> can then be used in a media feature. It **must** be used in a boolean context; using them in a normal or range context is a syntax error. The custom media query evaluates to true if the <media-query-list> it represents evaluates to true, and false otherwise.

#### EXAMPLE 49

For example, if a responsive site uses a particular breakpoint in several places, it can alias that with a reasonable name:

```
@custom-media --narrow-window (max-width: 30em);

@media (--narrow-window) {
  /* narrow window styles */
}

@media (--narrow-window) and (script) {
  /* special styles for when script is allowed */
}

/* etc */
```

## § 11.1 Script-based Custom Media Queries

**ISSUE 13** Define a map of names to values for JS. Values can be either a `MediaQueryList` object or a boolean, in which case it's treated identically to the above, or can be a number or a string, in which case it's treated like a normal MQ, and can use the normal or range context syntax. Like:

```
<script>
CSS.customMedia.set('--foo', 5);
</script>
<style>
@media (_foo: 5) { ... }
@media (_foo < 10) { ... }
</style>
```

## § 11.2 CSSOM

The `CSSRule` interface is extended as follows:

```
partial interface CSSRule {
  const unsigned short CUSTOM_MEDIA_RULE = 17;
};
```

The `CSSCustomMediaRule` interface represents a '@custom-media' rule.

```
interface CSSCustomMediaRule : CSSRule {
  attribute DOMString name;
  [SameObject, PutForwards=mediaText] readonly attribute MediaList media;
};
```

### **name, of type DOMString**

The `name` attribute on getting must return a `DOMString` object that contains the serialization of the `<extension-name>` defined for the associated rule.

On setting the `name` attribute, run the following steps:

1. Parse a component value from the value.
2. If the returned value is an `<extension-name>`, replace the associated rule's name with the `<extension-name>`'s representation.
3. Otherwise, do nothing.

***media*, of type MediaList, readonly**

The media attribute must return a MediaList object for the <media-query-list> specified with the associated rule.

## § Changes

### § Changes Since the 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 'scripting', 'pointer', 'hover', 'light-level', 'update-frequency', '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.)
- White space is required around the keyword "and" as well as after "not" and "only".
- All media types except for 'screen', 'print', 'speech', and 'all' are deprecated.

## § Acknowledgments

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

Comments from Arve Bersvendsen, Björn Höhrmann, Chris Lilley, Christoph Päper, L. David Baron, Erika J. Etemad, François Remy, Melinda Grant, Nicholas C. Zakas Philipp Hoschka, Rick Byers, Rijk van Geijtenbeek, Roger Gimson, Sigurd Lerstad, Simon Kissane, Simon Pieters, Steven Pemberton, and Susan Lesch improved this specification.

## § 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 50

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.

## § Experimental implementations

To avoid clashes with future CSS features, the CSS2.1 specification reserves a prefixed syntax for proprietary and experimental extensions to CSS.

Prior to a specification reaching the Candidate Recommendation stage in the W3C process, all implementations of a CSS feature are considered experimental. The CSS Working Group recommends that implementations use a vendor-prefixed syntax for such features, including those in W3C Working Drafts. This avoids incompatibilities with future changes in

the draft.

## § 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](mailto:public-css-testsuite@w3.org) mailing list.

## § References

### § Normative References

#### ¶ [CSS21]

Bert Bos; et al. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. 7 June 2011. W3C Recommendation. URL: <http://www.w3.org/TR/2011/REC-CSS2-20110607>

#### ¶ [CSS3SYN]

Tab Atkins Jr.; Simon Sapin. CSS Syntax Module. 5 November 2013. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/2013/WD-css-syntax-3-20131105/>

#### ¶ [CSS3VAL]

Håkon Wium Lie; Tab Atkins; Erika J. Etemad. CSS Values and Units Module Level 3. 30 July 2013. W3C Candidate Recommendation. (Work in progress.) URL: <http://www.w3.org/TR/2013/CR-css3-values-20130730/>

## ¶ [MEDIAQ]

Florian Rivoal. Media Queries. 19 June 2012. W3C Recommendation. URL: <http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/>

## ¶ [RFC2119]

S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. URL: <http://www.ietf.org/rfc/rfc2119.txt>

## § Informative References

### ¶ [CSS-DEVICE-ADAPT]

Rune Lillesveen. CSS Device Adaptation. 15 September 2011. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/2011/WD-css-device-adapt-20110915/>

### ¶ [HTML401]

Dave Raggett; Arnaud Le Hors; Ian Jacobs. HTML 4.01 Specification. 24 December 1999. W3C Recommendation. URL: <http://www.w3.org/TR/1999/REC-html401-19991224>

### ¶ [RFC2531]

G. Klyne; L. McIntyre. Content Feature Schema for Internet Fax. March 1999. URL: <http://www.ietf.org/rfc/rfc2531.txt>

### ¶ [XMLSTYLE]

James Clark. Associating Style Sheets with XML documents. 29 June 1999. W3C Recommendation. URL: <http://www.w3.org/1999/06/REC-xml-stylesheet-19990629>

## § Index

all, [2.3](#)

any-hover, [7.3](#)

any-pointer, [7.3](#)

aspect-ratio, [4.3](#)

aural, [2.3](#)

boolean context, [2.4.2](#)



braille, [2.3](#)

coarse, [7.1](#)

color, [6.1](#)

color-index, [6.2](#)

CSSCustomMediaRule, [11.2](#)

@custom-media, [11](#)

custom media query, [11](#)

CUSTOM\_MEDIA\_RULE, [11.2](#)

device-aspect-ratio, [4.7](#)

device-height, [4.6](#)

device-width, [4.5](#)

dim, [8.1](#)

<dimension>, [3](#)

embossed, [2.3](#)

enabled, [9.1](#)

fine, [7.1](#)

<general-enclosed>, [3](#)

grid, [5.3](#)

handheld, [2.3](#)

height, [4.2](#)

hover

- descriptor for @media, [7.2](#)
- value for @media/hover, [7.2](#)

<ident>, [3](#)

initial-only, [9.1](#)

interlace, [5.2](#)

light-level, [8.1](#)

media, [11.2](#)

<media-and>, [3](#)

<media-condition>, [3](#)

`<media-feature>`, [3](#)

media feature, [2.4](#)

`<media-in-parens>`, [3](#)

`<media-not>`, [3](#)

`<media-or>`, [3](#)

media query, [2](#)

`<media-query>`, [3](#)

media query list, [2.1](#)

`<media-query-list>`, [3](#)

media query modifier, [2.2](#)

media type, [2.3](#)

`<media-type>`, [3](#)

`<mf-boolean>`, [3](#)

`<mf-name>`, [3](#)

`<mf-plain>`, [3](#)

`<mf-range>`, [3](#)

`<mf-value>`, [3](#)

monochrome, [6.3](#)

`<mq-boolean>`, [1.2](#)

name, [11.2](#)

none

value for `@media/update-frequency`, [5.4](#)

value for `@media/overflow-block`, [5.5](#)

value for `@media/overflow-inline`, [5.6](#)

value for `@media/pointer`, [7.1](#)

value for `@media/hover`, [7.2](#)

value for `@media/scripting`, [9.1](#)

normal

value for `@media/update-frequency`, [5.4](#)

value for `@media/light-level`, [8.1](#)

not, [2.2.1](#)

on-demand, [7.2](#)

only, [2.2.2](#)

optional-paged, [5.5](#)

orientation, [4.4](#)

overflow-block, [5.5](#)

overflow-inline, [5.6](#)

paged, [5.5](#)

pointer, [7.1](#)

print, [2.3](#)

progressive, [5.2](#)

projection, [2.3](#)

range context, [2.4.3](#)

<ratio>, [1.2](#)

resolution, [5.1](#)

scan, [5.2](#)

screen, [2.3](#)

scripting, [9.1](#)

scroll

value for @media/overflow-block, [5.5](#)

value for @media/overflow-inline, [5.6](#)

slow, [5.4](#)

speech, [2.3](#)

tty, [2.3](#)

tv, [2.3](#)

update-frequency, [5.4](#)

washed, [8.1](#)

width, [4.1](#)

## § [Property index](#)

No properties defined.

## § '@media' Descriptors

Name	Value	Initial	Type
<u>'width'</u>	<length>		range
<u>'height'</u>	<length>		range
<u>'aspect-ratio'</u>	<ratio>		range
<u>'orientation'</u>	portrait   landscape		discrete
<u>'device-width'</u>	<length>		range
<u>'device-height'</u>	<length>		range
<u>'device-aspect-ratio'</u>	<ratio>		range
<u>'resolution'</u>	<resolution>		range
<u>'scan'</u>	interlace   progressive		discrete
<u>'grid'</u>	<mq-boolean>		discrete
<u>'update-frequency'</u>	none   slow   normal		discrete
<u>'overflow-block'</u>	none   scroll   optional-paged   paged		discrete
<u>'overflow-inline'</u>	none   scroll		discrete
<u>'color'</u>	<integer>		range
<u>'color-index'</u>	<integer>		range
<u>'monochrome'</u>	<integer>		range
<u>'pointer'</u>	none   coarse   fine		discrete
<u>'hover'</u>	none   on-demand   hover		discrete
<u>'any-pointer'</u>	none   coarse   fine		discrete
<u>'any-hover'</u>	none   on-demand   hover		discrete
<u>'light-level'</u>	dim   normal   washed		discrete
<u>'scripting'</u>	none   initial-only   enabled		discrete

## § Issues Index

**ISSUE 1** Reasonable to restrict `<ratio>` to `<integer>s`? I've seen aspect ratios written with decimal points in real life. ↵

**ISSUE 2** We should deprecate the `device-*` media features. They don't do anything useful, and are mostly just abused as a ghetto "phone versus desktop" media type. We've now defined a number of useful MQs for differentiating devices in useful ways instead. ↵

**ISSUE 3** Figure out how to make the above work properly for `</>` syntax. Just translate it over directly? That prevents you from doing a "less than/greater than" dichotomy without using `'not'`. Hmm. ↵

**ISSUE 4** Another media feature should probably be added to deal with the type of resolution authors want to know to deal with monochrome printing. ↵

**ISSUE 5** "Viewport" isn't the right term here, or in `'overflow-inline'`. ↵

**ISSUE 6** The names of `'overflow-block'` and `'overflow-inline'` are possibly confusing. Any better names? ↵

**ISSUE 7** Using this media feature for accessibility purposes overlaps a lot with the high-contrast media feature proposed by Microsoft. Can we adjust this so that it covers all use cases for both, or somehow modify them to work in an orthogonal, rather than overlapping, fashion? Also, the high-contrast media feature could be extended to also cover inverted colors, as discussed in <http://lists.w3.org/Archives/Public/www-style/2013Oct/0672.html> ↵

**ISSUE 8** Is there a use-case for distinguishing between "UA doesn't support scripting" and "scripting is supported, but turned off"? ↵

**ISSUE 9** We need a media feature (or set of media features) to detect the type of keyboard available. It should be able to distinguish between full computer keyboards, phone dial pads, tv remotes, or virtual keyboards. As an attempt at an exhaustive list is likely to fail, finding atomic features to decompose these into would be preferable, but these remain to be identified.

- always vs in text forms only
- just numbers vs free alphanumeric input vs full ime support
- work properly vs horrible lag like on a tv remote

How much is actually useful for styling?

←

**ISSUE 10** Example sets of MQs that would match on different types of devices

- printer: update-frequency:none, pointer:none, hover:none, overflow-block: paged, overflow-inline: none
- eink with stylus: update-frequency:slow, pointer:fine, hover:none, overflow-block: paged, overflow-inline: none
- tv: update-frequency: normal, pointer: none, hover: none: overflow-block: none, overflow-inline: none
- tablet: update-frequency: normal, pointer coarse, hover none, overflow-block: scroll, overflow-inline: scroll
- wii: update-frequency normal, pointer: coarse, hover: yes, overflow-block: scroll, overflow-inline: scroll
- Chromebook pixel: update-frequency: normal, pointer: coarse \*and\* fine, hover: over, overflow-block: scroll, overflow-inline: scroll
- Glass: update-frequency: normal, pointer: coarse, hover: none, overflow-block: none, overflow-inline: none
- XTERM: update-frequency: normal, pointer:none, hover:none, grid:1, overflow-block: scroll, overflow-inline: none

←

**ISSUE 11** <http://lists.w3.org/Archives/Public/www-style/2013Mar/0448.html>

MQ for detecting if the device is willing to display/print backgrounds and other ink-hungry properties.

↩

**ISSUE 12** What to do about the 'view-mode' media feature? ↩

**ISSUE 13** Define a map of names to values for JS. Values can be either a MediaQueryList object or a boolean, in which case it's treated identically to the above, or can be a number or a string, in which case it's treated like a normal MQ, and can use the normal or range context syntax. Like:

```
<script>
CSS.customMedia.set('--foo', 5);
</script>
<style>
@media (_foo: 5) { ... }
@media (_foo < 10) { ... }
</style>
```

↩