Media Queries Notes

<https://www.w3schools.com/css/css3_mediaqueries.asp>

CSS Media Queries

The @media rule, introduced in CSS2, made it possible to define different style rules for different media types.

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

* width and height of the viewport
* orientation of the viewport (landscape or portrait)
* resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

CSS Media Types

|  |  |
| --- | --- |
| **Value** | **Description** |
| all | Used for all media type devices |
| print | Used for print preview mode |
| screen | Used for computer screens, tablets, smart-phones etc. |

CSS Common Media Features

Here are some commonly used media features:

|  |  |
| --- | --- |
| **Value** | **Description** |
| orientation | Orientation of the viewport. Landscape or portrait |
| max-height | Maximum height of the viewport |
| min-height | Minimum height of the viewport |
| height | Height of the viewport (including scrollbar) |
| max-width | Maximum width of the viewport |
| min-width | Minimum width of the viewport |
| width | Width of the viewport (including scrollbar) |

Media Query Syntax

A media query consists of a media type and can contain one or more media features, which resolve to either true or false.

@media not|only *mediatype*and(*media feature*) and(*media feature*) { *CSS-Code;*}

The *mediatype* is optional (if omitted, it will be set to all). However, if you use **not** or **only**, you must also specify a *mediatype*.

The result of the query is true if the specified media type matches the type of device the document is being displayed on and all media features in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.

Meaning of the **not**, **only,** and **and** keywords:

**not:** This keyword inverts the meaning of an entire media query.

**only:** This keyword prevents older browsers that do not support media queries from applying the specified styles. **It has no effect on modern browsers.**

**and:** This keyword combines a media type and one or more media features.

You can also link to different stylesheets for different media and different widths of the browser window (viewport):

<link rel="stylesheet" media="print" href="print.css">  
<link rel="stylesheet" media="screen" href="screen.css">  
<link rel="stylesheet" media="screen and (min-width: 480px)" href="example1.css">  
<link rel="stylesheet" media="screen and (min-width: 701px) and (max-width: 900px)" href="example2.css">  
etc....

Media Queries Simple Examples

One way to use media queries is to have an alternate CSS section right inside your style sheet.

The following example changes the background-color to lightgreen if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the background-color will be pink):

Example

@media screen and (min-width: 480px) {  
  body {  
    background-color: lightgreen;  
  }  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss3_media_queries1)

The following example shows a menu that will float to the left of the page if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the menu will be on top of the content):

Example

@media screen and (min-width: 480px) {  
  #leftsidebar {width: 200px; float: left;}  
  #main {margin-left: 216px;}  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss3_media_queries2)

More Media Query Examples

For much more examples on media queries, go to the next page: [CSS MQ Examples](https://www.w3schools.com/css/css3_mediaqueries_ex.asp).

CSS @media Reference

For a full overview of all the media types and features/expressions, please look at the [@media rule in our CSS reference](https://www.w3schools.com/cssref/css3_pr_mediaquery.asp).

<https://css-tricks.com/a-complete-guide-to-css-media-queries/>

**A Complete Guide to CSS Media Queries**

[](https://css-tricks.com/author/agalante/)

[**Andrés Galante**](https://css-tricks.com/author/agalante/)on Oct 2, 2020 (Updated on Oct 19, 2022)

Media queries can modify the appearance (and even behavior) or a website or app based on a matched set of conditions about the user’s device, browser or system settings.

CSS Media queries are a way to target browser by certain characteristics, features, and user preferences, then apply styles or run other code based on those things. Perhaps the most common media queries in the world are those that target particular viewport ranges and apply custom styles, which birthed the whole idea of responsive design.

*/\* When the browser is at least 600px and above \*/*

@media screen and (min-width: 600px) {

.element {

*/\* Apply some styles \*/*

}

}

There are lots of other things we can target beside viewport width. That might be screen resolution, device orientation, operating system preference, or even more among a whole bevy of things we can query and use to style content.

Looking for a quick list of media queries based on the viewports of standard devices, like phones, tablets and laptops? [Check out our collection of snippets.](https://css-tricks.com/snippets/css/media-queries-for-standard-devices/)

**Using media queries**

Media queries are commonly associated with CSS, but they can be used in HTML and JavaScript as well.

HTML

CSS

JavaScript

**Anatomy of a Media Query**

Now that we’ve seen several examples of *where* media queries can be used, let’s pick them apart and see what they’re actually doing.



@media

@media [media-type] ([media-feature]) {

*/\* Styles! \*/*

}

The first ingredient in a media query recipe is the @media rule itself, which is one of [many CSS at-rules](https://css-tricks.com/the-at-rules-of-css/). Why does @media get all the attention? Because it’s geared to the *type* of media that a site is viewed with, what *features* that media type supports, and *operators* that can be combined to mix and match simple and complex conditions alike.

Media types

@media screen {

*/\* Styles! \*/*

}

What type of media are we trying to target? In many (if not most) cases, you’ll see a screen value used here, which makes sense since many of the media types we’re trying to match are devices with screens attached to them.

But screens aren’t the only type of media we can target, of course. We have a few, including:

* all: Matches all devices
* print: Matches documents that are viewed in a print preview or any media that breaks the content up into pages intended to print.
* screen: Matches devices with a screen
* speech: Matches devices that read the content audibly, such as a screenreader. This replaces the now deprecated [aural](https://drafts.csswg.org/mediaqueries-3/#background) type since [Media Queries Level 4](https://drafts.csswg.org/mediaqueries/#media-types).
* To preview print styles in a screen all major browsers can [emulate the output of a print stylesheet using DevTools](https://css-tricks.com/can-you-view-print-stylesheets-applied-directly-in-the-browser/). Other media types such as tty, tv,  projection,  handheld, braille, embossed and aural have been deprecated and, while [the spec continues to advise browsers to recognize them](https://www.w3.org/TR/mediaqueries-4/#media-types), they must evaluate to nothing. If you are using one of these consider changing it for a modern approach.

Media features

Once we define the type of media we’re trying to match, we can start defining what features we are trying to match it to. We’ve looked at a lot of examples that match screens to width, where screen is the *type* and bothmin-width and max-width are *features* with specific values.

But there are many, many (many!) more “features” we can match. [Media Queries Level 4](https://www.w3.org/TR/mediaqueries-4/#mq-features) groups 18 media features into 5 categories.

**Viewport/Page Characteristics**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| width | Defines the widths of the viewport. This can be a specific number (e.g. 400px) or a range (using min-width and max-width). | <length> |  |
| height | Defines the height of the viewport. This can be a specific number (e.g. 400px) or a range (using min-height and max-height). | <length> |  |
| aspect-ratio | Defines the width-to-height aspect ratio of the viewport | <ratio> |  |
| orientation | The way the screen is oriented, such as tall (portrait) or wide (landscape) based on how the device is rotated. | portrait  landscape |  |
| overflow-block | Checks how the device treats content that overflows the viewport in the block direction, which can be scroll (allows scrolling), optional-paged (allows scrolling and manual page breaks), paged (broken up into pages), and none (not displayed). | scroll  optional-paged  paged | Media Queries Level 4 |
| overflow-inline | Checks if content that overflows the viewport along the inline axis be scrolled, which is either none (no scrolling) or scroll (allows scrolling). | scroll  none | Media Queries Level 4 |

**Display Quality**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| resolution | Defines the target pixel density of the device | <resolution>  infinite |  |
| scan | Defines the scanning process of the device, which is the way the device paints an image onto the screen (where interlace draws odd and even lines alternately, and progressive draws them all in sequence). | interlace  progressive |  |
| grid | Determines if the device uses a grid (1) or bitmap (0) screen | 0 = Bitmap 1 = Grid | Media Queries Level 5 |
| update | Checks how frequently the device can modify the appearance of content (if it can at all), with values including none, slow and fast. | slow  fast  none | Media Queries Level 4 |
| environment-blending | A method for determining the external environment of a device, such as dim or excessively bright places. | opaque  additive  subtractive |  |
| display-mode | Tests the display mode of a device, including fullscreen(no browsers chrome), standalone (a standalone application), minimal-ui (a standalone application, but with some navigation), and browser (a more traditional browser window) | fullscreen  standalone  minimal-ui  browser | [Web App Manifest](https://w3c.github.io/manifest/#the-display-mode-media-feature) |

**Color**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| color | Defines the color support of a device, expressed numerically as bits. So, a value of 12 would be the equivalent of a device that supports 12-bit color, and a value of zero indicates no color support. | <integer> |  |
| color-index | Defines the number of values the device supports. This can be a specific number (e.g. 10000) or a range (e.g. min-color-index: 10000, max-color-index: 15000), just like width. | <integer> |  |
| monochrome | The number of bits per pixel that a device’s monochrome supports, where zero is no monochrome support. | <integer> |  |
| color-gamut | Defines the range of colors supported by the browser and device, which could be srgb, p3 or rec2020 | srgb  p3  rec2020 | Media Queries Level 4 |
| dynamic-range | The combination of how much brightness, color depth, and contrast ratio supported by the video plane of the browser and user device. | standard  high |  |
| inverted-colors | Checks if the browser or operating system is set to invert colors (which can be useful for optimizing accessibility for sight impairments involving color) | inverted  none | Media Queries Level 5 |

**Interaction**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| pointer | Sort of like any-pointer but checks if the *primary* input mechanism is a pointer and, if so, how accurate it is (where coarse is less accurate, fine is more accurate, and none is no pointer). | coarse  fine  none | Media Queries Level 4 |
| hover | Sort of like any-hover but checks if the *primary* input mechanism (e.g. mouse of touch) allows the user to hover over elements | hover  none | Media Queries Level 4 |
| any-pointer | Checks if the device uses a pointer, such as a mouse or styles, as well as how accurate it is (where coarse is less accurate and fine is more accurate) | coarse  fine  none | Media Queries Level 4 |
| any-hover | Checks if the device is capable of hovering elements, like with a mouse or stylus. In some rare cases, touch devices are capable of hovers. | hover  none | Media Queries Level 4 |

**Video Prefixed**

The spec references user agents, including TVs, that render video and graphics in two separate planes that each have their own characteristics. The following features describe those planes.

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| video-color-gamut | Describes the approximate range of colors supported by the video plane of the browser and user device | srgb  p3  rec2020 | Media Queries Level 5 |
| video-dynamic-range | The combination of how much brightness, color depth, and contrast ratio supported by the video plane of the browser and user device. | standard  high | Media Queries Level 5 |
| video-width¹ | The width of the video plane area of the targeted display | <length> | Media Queries Level 5 |
| video-height¹ | The height of the video plane area of the targeted display | <length> | Media Queries Level 5 |
| video-resolution¹ | The resolution of the video plane area of the targeted display | <resolution>  inifinite | Media Queries Level 5 |

¹ Under discussion (Issue [#5044](https://github.com/w3c/csswg-drafts/issues/5044))

**Scripting**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| scripting | Checks whether the device allows scripting (i.e. JavaScript) where enabled allows scripting, iniital-only | enabled  initial-only | Media Queries Level 5 |

**User Preference**

| **Feature** | **Summary** | **Values** | **Added** |
| --- | --- | --- | --- |
| prefers-reduced-motion | Detects if the user’s system settings are set to reduce motion on the page, which is a [great accessibility check](https://css-tricks.com/introduction-reduced-motion-media-query/). | no-preference  reduce | Media Queries Level 5 |
| prefers-reduced-transparency | Detects if the user’s system settings prevent transparent across elements. | no-preference  reduce | Media Queries Level 5 |
| prefers-contrast | Detects if the user’s system settings are set to either increase or decrease the amount of contrast between colors. | no-preference  high  low  forced | Media Queries Level 5 |
| prefers-color-scheme | Detects if the user prefers a light or dark color scheme, which is a rapidly growing way to go about [creating “dark mode” interfaces](https://css-tricks.com/a-complete-guide-to-dark-mode-on-the-web/). | light  dark | Media Queries Level 5 |
| forced-colors | Tests whether the browser restricts the colors available to use (which is none or active) | active  none | Media Queries Level 5 |
| prefers-reduced-data | Detects if the user prefers to use less data for the page to be rendered. | no-preference  reduce | Media Queries Level 5 |

**Deprecated**

| **Name** | **Summary** | **Removed** |
| --- | --- | --- |
| device-aspect-ratio | The width-to-height aspect ratio of the output device | Media Queries Level 4 |
| device-height | The height of the device’s surface that displays rendered elements | Media Queries Level 4 |
| device-width | The width of the device’s surface that displays rendered elements | Media Queries Level 4 |

Operators

Media queries support logical operators like many programming languages so that we can match media types based on certain conditions. The @media rule is itself a logical operator that is basically stating that “if” the following types and features are matches, then do some stuff.

**and**

But we can use the and operator if we want to target screens within a range of widths:

*/\* Matches screen between 320px AND 768px \*/*

@media screen (min-width: 320px) and (max-width: 768px) {

.element {

*/\* Styles! \*/*

}

}

**or (or comma-separated)**

We can also comma-separate features as a way of using an or operator to match different ones:

*/\**

*Matches screens where either the user prefers dark mode or the screen is at least 1200px wide \*/*

@media screen (prefers-color-scheme: dark), (min-width 1200px) {

.element {

*/\* Styles! \*/*

}

}

**not**

Perhaps we want to target devices by what they do **not** support or match. This declaration removes the body’s background color when the device is a printer and can only show one color.

@media print and ( not(color) ) {

body {

background-color: none;

}

}

Want to go deeper? Check out [“CSS Media Queries: Quick Reference & Guide”](https://www.digitalocean.com/community/tutorials/css-media-queries?utm_medium=content_acq&utm_source=css-tricks&utm_campaign=&utm_content=awareness_bestsellers) from the DigitalOcean community for more examples that follow the syntax for media quieries.

**Do you really need CSS media queries?**

Media queries are a powerful tool in your CSS toolbox with exciting hidden gems. But if you accomodate your design to every possible situation you’ll end up with a codebase that’s too complex to maintain and, as we all know, CSS is like a bear cub: cute and inoffensive but when it grows it will eat you alive.

That’s why I recommend following Ranald Mace’s concept of Universal Design which is “*the design of products to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design.”*

In [“Accessibility for Everyone”](https://abookapart.com/products/accessibility-for-everyone) Laura Kalbag explains that the difference between accessible and universal design is subtle but important. An accessible designer would create a large door for people on a wheel chair to enter, while a universal designer would produce an entry that anyone would fit disregarding of their abilities.

I know that talking about universal design on the web is hard and almost sound utopian, but think about it, there are around [150 different browsers](https://en.wikipedia.org/wiki/List_of_web_browsers), around 50 different combinations of user preferences, and as we mentioned before more than 24000 different and unique Android devices alone.

This means that there are at least 18 million possible cases in which your content might be displayed. In the words of the fantastic [Miriam Suzanne](https://twitter.com/MiriSuzanne/status/1261075490628005888), *“CSS out here trying to do graphic design of unknown content on an infinite and unknown canvas, across operating systems, interfaces, & languages. There’s no possible way for any of us to know what we’re doing.”*

That’s why assuming is really dangerous, so when you design, develop and think about your products leave assumptions behind and use media queries to make sure that your content is displayed correctly in any contact and before any user.

**Matching value ranges**

Many of the media features outlined in the previous section — including width, height, color and color-index — can be prefixed with min- or max- to express minimum or maximum constraints. We’ve already seen these in use throughout many of the examples, but the point is that we can create a range of value to match instead of having to declare specific values.

In the following snippet, we’re painting the body’s background purple when the viewport width is wider than 30em and narrower than 80em. If the viewport width does not match that range of values, then it will fallback to white.

body {

background-color: #fff;

}

@media (min-width: 30em) and (max-width: 80em) {

body {

background-color: purple;

}

}

Media Queries Level 4 specifies a [new and simpler syntax](https://www.w3.org/TR/mediaqueries-4/#mq-ranges) using less then (<), greater than (>) and equals (=) operators. So, that last example can be converted to the new syntax, like so:

@media (30em <= width <= 80em) {

*/\* ... \*/*

}

**Nesting and complex decision making**

CSS allows you to nest at-rules or group statements using parentheses, making it possible to go as deep as we want to evaluate complex operations.

@media (min-width: 20em), not all and (min-height: 40em) {

@media not all and (pointer: none) { ... }

@media screen and ( (min-width: 50em) and (orientation: landscape) ), print and ( not (color) ) { ... }

}

**Be careful!**even thought it’s possible to create powerful and complex expressions, you might end up with a very opinionated, hard to maintain query. As Brad Frost [puts it](https://bradfrost.com/blog/post/7-habits-of-highly-effective-media-queries/): “The more complex our interfaces are, the more we have to think to maintain them properly.”

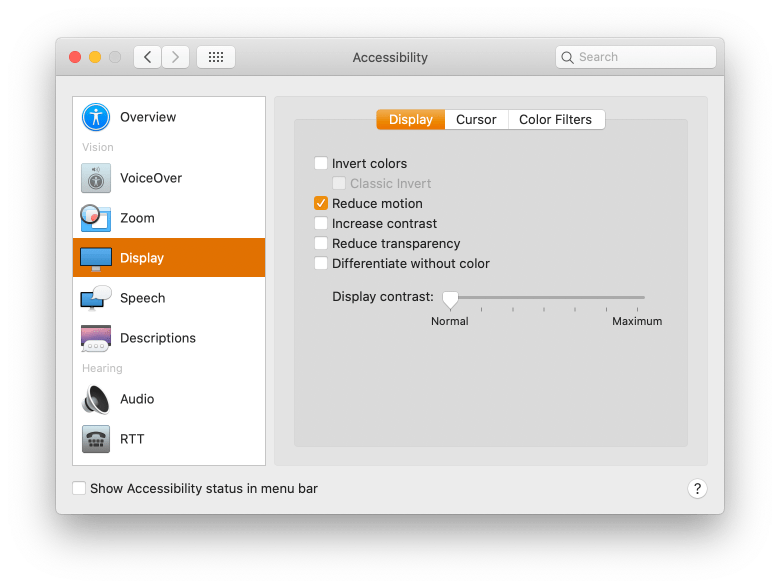
**Accessibility**

Many of the features added in Media Queries Level 4 are centered around accessibility.

**prefers-reduced-motion**

prefers-reduced-motion detects if the user has the reduced motion preference activated to minimize the amount of movements and animations. It takes two values:

* no-preference: Indicates that the user has made no preference known to the system.
* reduce: Indicates that user has notified the system that they prefer an interface that minimizes the amount of movement or animation, preferably to the point where all non-essential movement is removed.



This preference is generally used by people who suffer from vestibular disorder or vertigo, where different movements result in loss of balance, migraine, nausea or hearing loss. If you ever tried to spin quickly and got dizzy, you know what it feels like.

In a [fantastic article](https://css-tricks.com/revisiting-prefers-reduced-motion-the-reduced-motion-media-query/) by Eric Bailey, he suggests stopping all animations with this code:

@media screen and (prefers-reduced-motion: reduce) {

\* {

*/\* Very short durations means JavaScript that relies on events still works \*/*

animation-duration: 0.001ms !important;

animation-iteration-count: 1 !important;

transition-duration: 0.001ms !important;

}

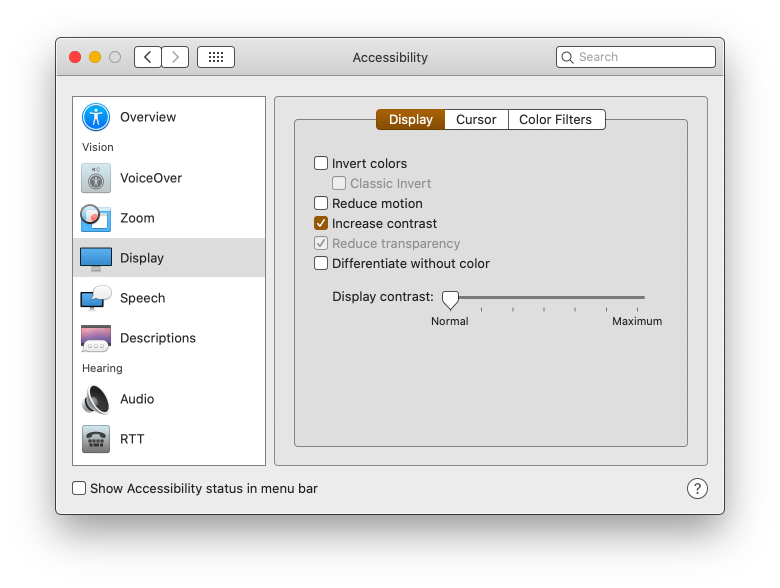
}

Popular frameworks like [Bootstrap have this feature on by default](https://v5.getbootstrap.com/docs/5.0/getting-started/accessibility/#reduced-motion). In my opinion there is no excuse not to use prefers-reduced-motion — just use it.

**prefers-contrast**

The prefers-contrast feature informs whether the user has chosen to increase or reduce contrast in their system preferences or the browser settings. It takes three values:

* no-preference: When a user has made no preference known to the system. If you use it as a boolean it’ll evaluate false.
* high: When a user has selected the option to display a higher level of contrast.
* low: When a user has selected the option to display a lower level of contrast.



At the moment of writing this feature is not supported by any browser. Microsoft has done a [non-standard earlier implementation](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/-ms-high-contrast) with the -ms-high-contrast feature that works only on Microsoft Edge v18 or earlier (but not Chromium-based versions).

.button {

background-color: #0958d8;

color: #fff;

}

@media (prefers-contrast: high) {

.button {

background-color: #0a0db7;

}

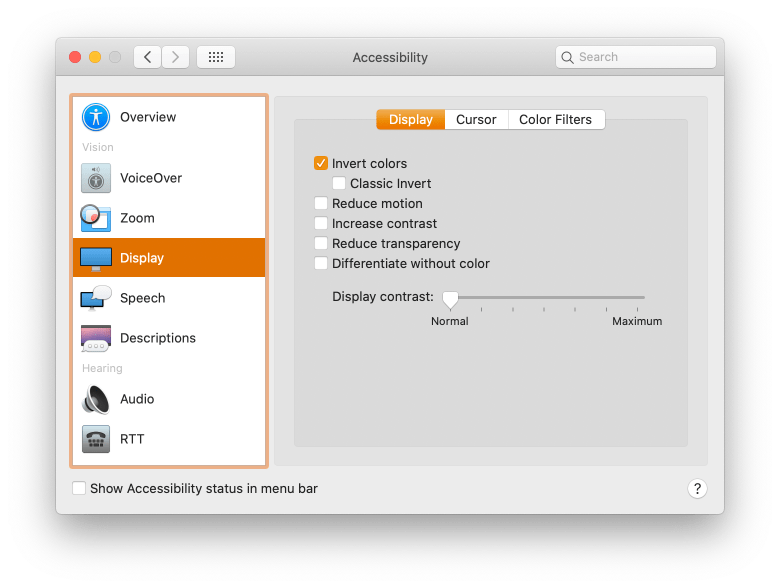
}

This example is increasing the contrast of a the class button from AA to AAA when the user has high contrast on.

**inverted-colors**

The inverted-colors feature informs whether the user has chosen to invert the colors on their system preferences or the browser settings. Sometimes this option is used as an alternative to high contrast. It takes three values:

* none: When colors are displayed normally
* inverted: When a user has selected the option to invert colors



The problem with inverted colors is that it’ll also invert the colors of images and videos, making them look like x-ray images. By using a CSS invert filter you can select all images and videos and invert them back.

@media (inverted-colors) {

img, video {

filter: invert(100%);

}

}

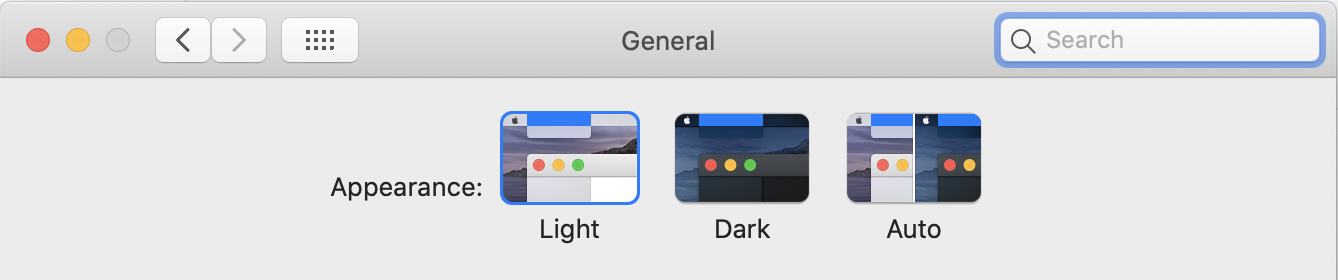
At the time of writing this feature is only supported by Safari.

**prefers-color-scheme**

Having a “dark mode” color scheme is something we’re seeing a lot more of these days, and thanks to the prefers-color-scheme feature, we can tap into a user’s system or browser preferences to determine whether we serve a “dark” or a “light” theme based on the ir preferences.

It takes two values:

* light: When a user has selected that they prefer a light theme or has no active preferences
* dark: When a user has selected a dark display in their settings



body {

--bg-color: white;

--text-color: black;

background-color: var(--bg-color);

color: var(--text-color);

}

@media screen and (prefers-color-scheme: dark) {

body {

--bg-color: black;

--text-color: white;

}

}

As Adhuham explains in the [complete guide to Dark Mode](https://css-tricks.com/a-complete-guide-to-dark-mode-on-the-web/) there is way more to it than just changing the color of the background. Before you jump into doing dark mode remember that if you don’t have a very smart implementation strategy you might end up with a code base that’s really hard to maintain. [CSS variables](https://css-tricks.com/theming-with-variables-globals-and-locals/) can do wonders for it but that’s a subject for another article.

**What lies ahead?**

[Media Queries Level 5](https://www.w3.org/TR/mediaqueries-5/) is currently in Working Draft status, which means a lot can change between now and when it becomes a recommendation. But it includes interesting features that are worth mentioning because they open up new ways to target screens and adapt designs to very specific conditions.

**User preference media features**

Hey, we just covered these in the last section! Oh well. These features are exciting because they’re informed by a user’s actual settings, whether they are from the user agent or even at the operating system level.

**Detecting a forced color palette**

This is neat. Some browsers will limit the number of available colors that can be used to render styles. This is called [“forced colors mode”](https://www.w3.org/TR/css-color-adjust-1/#forced-colors-mode) and, if enabled in the browser settings, the user can choose a limited set of colors to use on a page. As a result, the user is able to define color combinations and contrasts that make content more comfortable to read.

The [forced-colors](https://www.w3.org/TR/mediaqueries-5/#forced-colors) feature allows us to detect if a forced color palette is in use with the active value. If matched, the browser must provide the required color palette through the CSS system colors. The browser is also given the leeway to determine if the background color of the page is light or dark and, if appropriate, trigger the appropriate prefers-color-scheme value so we can adjust the page.

**Detecting the maximum brightness, color depth, and contrast ratio**

Some devices (and browsers) are capable of super bright displays, rendering a wide range of colors, and high contrast ratios between colors. We can detect those devices using the [dynamic-range](https://www.w3.org/TR/mediaqueries-5/#dynamic-range) feature, where the high keyword matches these devices and standard matches everything else.

We’re likely to see changes to this because, as of right now, there’s still uncertainty about what measurements constitute “high” levels of brightness and contrast. The browser may get to make that determination.

**Video prefixed features**

The spec talks about some screens, like TVs, that are capable of displaying video and graphics on separate “planes” which might be a way of distinguishing the video frame from other elements on the screen. As such, Media Queries Level 5 is proposing a new set of media features aimed at [detecting video characteristics](https://www.w3.org/TR/mediaqueries-5/#video-prefixed-features), including color gamut and dynamic range.

There are also proposals to detect video height, width and resolution, but [the jury’s still out](https://github.com/w3c/csswg-drafts/issues/5044) on whether those are the right ways to address video.

**Browser support**

Browsers keep evolving and since by the time you are reading this post chances are that browser support for this feature might change, please check [MDN updated browser compatibility table](https://developer.mozilla.org/en-US/docs/Web/CSS/@media).

**A note on container queries**

Wouldn’t be cool if components could adapt themselves on their own size instead of the browser’s? That’s what the concept of [CSS Container Queries](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Container_Queries) is all about. We currently only have the browser screen to make those changes via media queries.

That’s unfortunate, as the viewport isn’t always a direct relationship to how big the element itself is. Imagine a widget that renders in many different contexts on a site: sometimes in a sidebar, sometimes in a full-width footer, sometimes in a grid with unknown columns.

This is the problem that container queries try to solve. Ideally we could adapt styles of an element according to the size of itself instead of of the size of the viewport. Chrome 105 released support for CSS Container Queries. Same deal with Safari 16.1. Firefox is all we’re really waiting at the time of writing to get broad support.

This browser support data is from [Caniuse](http://caniuse.com/" \l "feat=css-container-queries), which has more detail. A number indicates that browser supports the feature at that version and up.

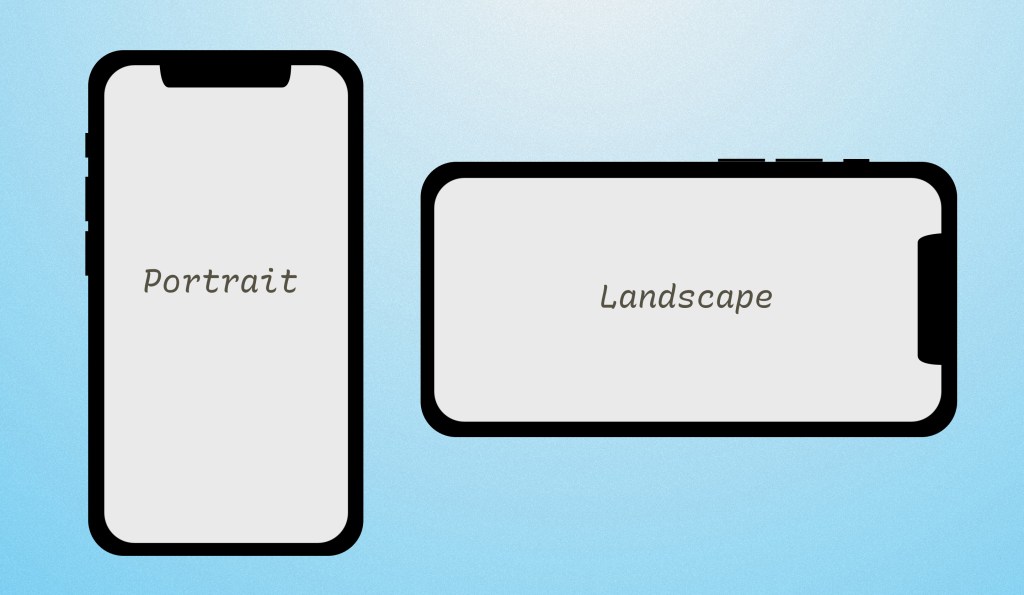
**Desktop**

| **Chrome** | **Firefox** | **IE** | **Edge** | **Safari** |
| --- | --- | --- | --- | --- |
| **106** | **110** | **No** | **106** | **16.0** |

**Mobile / Tablet**

| **Android Chrome** | **Android Firefox** | **Android** | **iOS Safari** |
| --- | --- | --- | --- |
| **121** | **122** | **121** | **16.0** |

**Examples**



**Specifications**

* [Media Queries Level 4](https://www.w3.org/TR/mediaqueries-4/) (Candidate Recommendation)
* [Media Queries Level 5](https://www.w3.org/TR/mediaqueries-5/) (Working Draft)