Use CSS media queries to create responsive websites

One design for all screen sizes

Jeff Bail

October 23, 2012

Static websites with fixed widths are quickly being replaced by responsive, fluid designs that scale up and down depending on screen size. With a responsive design you can present a usable interface no matter what device or screen is accessing the website. A responsive design also "future proofs" the website by responding to varying screen sizes, which are changing rapidly as new smart phones and tablets hit the market. The primary means of achieving a responsive design is with CSS media queries. In this article, learn to use media queries for desktop websites, mobile phones, and tablets.

Introduction

Every day, more phones and tablets come on the market. Consumers have a variety of devices in every size and shape imaginable, but website developers are left with a unique challenge: How to keep their website looking good for traditional browsers as well as mobile and tablet browsers. How do we provide a top-notch user experience on all of the different screen sizes? The answer: responsive design. A responsive design changes to fit the size of the screen on which it is displayed. The primary way to achieve a responsive design is by using CSS media queries.

In this article, explore how to use media queries for desktop websites, mobile phones, and tablets. Also learn to detect media query support and provide a fallback experience for older browsers that don't yet support this technique.

Prerequisites

The code samples in this article are designed to run in web browsers that support CSS media queries, including Mozilla Firefox, Apple Safari, Google Chrome, and Opera. See Related topics for a complete list showing browser compatibility support of CSS3 media queries in desktop and mobile browsers.

Responsive design

A responsive design changes to fit the size of the screen on which it is displayed; it doesn't look identical on every screen it's rendered upon. The responsive design renders an optimized version of the UI according to the screen properties it has available to use.
For example, if a website layout has a sidebar that takes up 25% of the screen width, this sidebar could appear drastically different on different screen sizes. It could be wide on a large desktop monitor and extremely narrow on a small smartphone screen. It's likely that at either the high end or low end of these screen sizes, the sidebar is no longer a usable UI component.

With media queries, you can write CSS that automatically changes the design to deliver an optimal UI experience for different screen sizes.

### Media queries

Media support has been available in CSS, through media types, since CSS version 2. If you've ever made a print style sheet, you've already used media types. Listing 1 shows an example.

#### Listing 1. Using media types

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

In Listing 1, the `media` attribute defines the style sheet that should be used for each type of media specified:

- `screen` is meant for color computer screens.
- `print` is intended for content that's viewed in print preview mode or printed on a printer.

Media queries, which are part of the CSS version 3 specification, extend the functions of media types and allow for more precise display rules in style sheets. A media query is an expression that evaluates to either True or False. If True, the styles it contains will be used. If False, they will not. This simple logic, made powerful through the use of expressions, gives you more flexibility in applying custom display rules for specific design scenarios.

A media query consists of one media type followed by one or more expressions that check for a particular condition, such as minimum screen width. Inside a style sheet, the media query rule might look like the example in Listing 2.

#### Listing 2. Media query rule

```css
@media all and (min-width: 800px) { ... }
```

According to the markup in Listing 2, for all types of media (screen, print, and so on) where the minimum horizontal screen width is 800 pixels, use the following CSS rules. The rules would appear where the ellipsis is in the example. For that media query:

- `@media all` is the media type, which says apply this CSS to all media types.
- `(min-width:800px)` is the expression that comprises the media query that tells the browser to apply only the following CSS if the browser has a minimum width of 800 pixels.

Note that in Listing 2 the keywords `all` and `and` can be omitted. When a media query applies to all media types, `all` can be omitted. The trailing `and` is also optional. Rewriting the media query to use shorthand syntax would look like Listing 3.
Listing 3. Shorthand syntax

```css
@media (min-width:800px) { ... }
```

Media queries can also contain complex expressions. For example, if you wanted to create a style that applied only when the minimum width was 800 pixels and the maximum width was 1200 pixels, you could do so with the rule in Listing 4.

Listing 4. Complex expression

```css
@media (min-width:800px) and (max-width:1200px) { ... }
```

You can have as many `and` conditions in your expression as you'd like. If you want to add another condition to check for a specific screen orientation, just add another `and` keyword followed by an additional `orientation` media query, as in Listing 5.

Listing 5. `and` conditions

```css
@media (min-width:800px) and (max-width:1200px) and (orientation:portrait) { ... }
```

The media query in Listing 5 is activated only when the minimum width is 800 pixels, the maximum width is 1200 pixels, and the orientation is portrait. (Orientation is typically only meaningful on smart phones and tablets that can easily be rotated between portrait and landscape modes.) If any of these conditions is False, the media query rule won't be applied.

The opposite of the `and` keyword is the `or` keyword. Like `and`, it groups conditionals together to form complex expressions. An expression with `or` separating two conditions will be True if either of the conditions is true, as in Listing 6.

Listing 6. `or` keyword

```css
@media (min-width:800px) or (orientation:portrait) { ... }
```

If the width is at least 800 pixels or the orientation is portrait, the rule will be applied.

Another media query keyword to keep in your arsenal is `not`. Placed at the beginning of a media query, `not` will negate the result. In other words, if the query would have been true without the `not` keyword, it will become false. Listing 7 shows an example.

Listing 7. Using `not`

```css
@media (not min-width:800px) { ... }
```

In plain English, the code in Listing 7 means that whenever the minimum width is not 800 pixels, apply the following CSS rules. The examples only use pixels as their unit of measurement inside the media query, but you aren't limited to pixels. You can use any valid CSS unit of measurement, such as centimeters (cm), inches (in), millimeters (mm), and so on.
Useful media features

There are several media features, such as width, color, and grid, that you can use in media queries. Descriptions of every possible media feature are outside the scope of this article. The World Wide Web Consortium’s (W3C) documentation on media queries has a complete list (see Related topics).

To design responsive websites, you need to know only a few media features: orientation, width, and height. Orientation, which is a simple media feature, can have a value of either portrait or landscape. These values match the way a user might hold the phone or tablet, which lets you optimize the content for both form factors. A screen is considered to be in portrait mode when the height exceeds the length, and in landscape mode when the width is greater than the height. Listing 8 shows an example of using an orientation media query.

Listing 8. orientation media query

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

Height and width behave similarly; both allow the min- and max- prefixes. Listing 9 shows a valid media query.

Listing 9. Height and width media query

```css
@media (min-width:800px) and (min-height:400px) { ... }
```

You can also use the width and height media features without either the min- or max- prefixes, as in Listing 10.

Listing 10. Without min- and max- prefixes

```css
@media (width:800px) and (height:400px) { ... }
```

The media query in Listing 10 would match when the screen is exactly 800 pixels wide and exactly 400 pixels in height. In the real world, a media query like this is probably too specific to be useful. Detecting the exact dimensions is a scenario that most, if not all, website visitors would likely never encounter. Typically, responsive design uses ranges to perform screen detection.

As a natural follow-on to media query size ranges, the next section explores some common media queries that you might find useful when designing a responsive site.

Common media queries

Because Apple was the first to market with consumer smart phone and tablet products, most of the following media queries are based on the size of those devices.

To target a smart phone in landscape mode, use:  
```
@media (min-width: 321px) { ... }
```

To target a smart phone in portrait mode, use: 
```
@media (max-width: 320px) { ... }
```
To target an Apple iPad in landscape mode, use:

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

To target an iPad in portrait mode, use:

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

You might have noticed use of the `orientation` media feature on the iPad and use of `width` on the Apple iPhone. Why? Unfortunately, the iPhone does not support the `orientation` media feature. You must use `width` to simulate these orientation breakpoints. See Related topics for more information about common media queries.

**Media queries in SASS**

With built-in support in Ruby on Rails helping to propel its popularity, Syntactically Awesome Style Sheets (SASS) is gaining traction in the web development community. A detailed discussion of SASS is outside the scope of this article, but I'll cover the basics of using media queries in your SASS-based style sheets. See Related topics to learn more about SASS.

Media queries in SASS behave exactly as they do in normal CSS, with one exception: They can be nested inside other CSS rules. When a media query is nested inside another CSS rule, it places the rule at the top level of the style sheet, as in Listing 11.

**Listing 11. Nested media query**

```scss
#header {
  width: 400px;
  @media (min-width: 800px) {
    width: 100%;
  }
}
```

The example in Listing 11 would compile into the code in Listing 12.

**Listing 12. Compiled result**

```scss
#header {
  width: 400px;
}
@media (min-width: 800px) {
  #header {
    width: 100%;
  }
}
```

**Organizing your media queries**

Now that you know how to write media queries, it’s time to think about deploying them into your CSS code base in a logical, organized way. Deciding how to organize your media queries is largely a matter of personal preference. This section explores the benefits and drawbacks of two main approaches.

The first approach is to specify entirely different style sheets for different screen sizes. The benefit is that the rules are kept in separate style sheets, which keeps the display logic cleanly separated and perhaps easier for a team to maintain. It might also make merging between source code
branches a bit easier. This benefit is also its downside. By creating separate style sheets for each media query, you no longer group all of an element’s styles in the same place in the same file. Your website's CSS could be more difficult to maintain by creating more places to look when altering the CSS for an element.

The second approach is to use media queries in your existing style sheets right next to where the rest of the element’s styles are defined. The benefit is that it keeps all element styling together in the same place. This practice can create more source code merge work when working in teams, but this is a manageable and common part of all team-based software development.

There is no right or wrong approach. Choose the approach that makes the most sense for your project and your team.

**Browser support**

By now you’re probably convinced that CSS media queries are a great tool and are wondering which browsers support CSS media queries. There's good news and bad news on this front.

- The good news: Most modern browsers (including those on smart phones) support CSS media queries.
- The bad news: As recently as Windows® Internet Explorer® 8, media queries weren't supported by the browser from Redmond. For most professional web developers, this means you need a solution to support media queries in Internet Explorer.

The solution below is a JavaScript polyfill called respond.js.

**Polyfill with respond.js**

Respond.js is a tiny JavaScript library that enhances web browsers and enables CSS media queries in browsers that don't natively support them. The script loops through all CSS referenced on the page and parses out the CSS rules that use media queries. The script then monitors browser width changes to append and remove the styles that match the media query in the CSS. The end result is working CSS media queries on browsers that don't natively support them.

Because this is a JavaScript-based solution, the browser will need to have JavaScript enabled for the script to work. The script has a narrow focus on supporting only the minimum and maximum width media queries needed to create responsive designs. It is not meant to be a full-scale replacement for every possible CSS media query out there. See Related topics to read more about the script's features and limitations.

Respond.js is one among many available open source media query polyfills you can choose from. If you find respond.js doesn't meet your needs, a little research will unveil several alternatives.

**Conclusion**

With CSS media queries, you can easily target specific screen sizes and create a solid user experience no matter what type of browser or device is used to visit your website. These
techniques are at the center of responsive design, which is an emerging mobile web design and development practice. There’s no real cost involved with trying out media queries with your website (other than simply enhancing your existing CSS files), so why not give them a try? Visitors browsing your website on tablets, phones, and e-readers will thank you.
Related topics

- **When Can I Use CSS3 Media Queries?**: See a list showing browser support for CSS media queries.
- **W3C requirement for media queries**: Get a full list of possible media features.
- **Media Queries for Standard Devices**: Get more information on common media queries.
- **SASS**: Find details on using SASS from its official site.
- **XML area on developerWorks**: Find the resources you need to advance your skills in the XML arena, including DTDs, schemas, and XSLT. See the XML technical library for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **XML technical library**: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **Respond.js**: Visit the official Respond.js project on Github.
- **developerWorks Web development zone**: Find articles covering various web-based solutions. See the Web development technical library for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **IBM product evaluation versions** and get your hands on application development tools and middleware products from DB2, Lotus, Rational, Tivoli, and WebSphere.