

Responsive Design

A large, white, stylized letter 'C' is positioned in the bottom right corner of the page. The background of the entire page is a dark green color with a network of white lines and nodes, and several overlapping light green circles of varying sizes.

Imprint

Copyright 2012 Smashing Media GmbH, Freiburg, Germany

Version 1: June 2012

ISBN: 978-3-943075-33-5

Cover Design: Ricardo Gimenes

PR & Press: Stephan Poppe

eBook Strategy: Thomas Burkert

Technical Editing: Talita Telma Stöckle, Andrew Rogerson

Idea & Concept: Smashing Media GmbH

ABOUT SMASHING MAGAZINE

[Smashing Magazine](#) is an online magazine dedicated to Web designers and developers worldwide. Its rigorous quality control and thorough editorial work has gathered a devoted community exceeding half a million subscribers, followers and fans. Each and every published article is carefully prepared, edited, reviewed and curated according to the high quality standards set in Smashing Magazine's own publishing policy. Smashing Magazine publishes articles on a daily basis with topics ranging from business, visual design, typography, front-end as well as back-end development, all the way to usability and user experience design. The magazine is — and always has been — a professional and independent online publication neither controlled nor influenced by any third parties, delivering content in the best interest of its readers. These guidelines are continually revised and updated to assure that the quality of the published content is never compromised.

ABOUT SMASHING MEDIA GMBH

[Smashing Media GmbH](#) is one of the world's leading online publishing companies in the field of Web design. Founded in 2009 by Sven Lennartz and Vitaly Friedman, the company's headquarters is situated in southern Germany, in the sunny city of Freiburg im Breisgau. Smashing Media's lead publication, Smashing Magazine, has gained worldwide attention since its emergence back in 2006, and is supported by the vast, global Smashing community and readership. Smashing Magazine had proven to be a

trustworthy online source containing high quality articles on progressive design and coding techniques as well as recent developments in the Web design industry.

About this eBook

This eBook, "Responsive Design", gives an overview about responsive Web design, showing many situations and techniques in which this approach can be applied to. Smartphones, tablets, laptops and desktop computers may share the same design, which is adapted according to screen size, platform and orientation of each device. Flexible grids and layouts, images, text and an intelligent use of CSS media queries are included.

Table of Contents

[Responsive Web Design: What It Is And How To Use It](#)

[Progressive And Responsive Navigation](#)

[Techniques For Gracefully Degrading Media Queries](#)

[Is There Ever A Justification For Responsive Text?](#)

[How To Use CSS3 Media Queries To Create A Mobile Version Of Your Website](#)

[Device-Agnostic Approach To Responsive Web Design](#)

[Content Prototyping In Responsive Web Design](#)

[About The Authors](#)

Responsive Web Design: What It Is And How To Use It

Kayla Knight

Almost every new client these days wants a mobile version of their website. It's practically essential after all: one design for the BlackBerry, another for the iPhone, the iPad, netbook, Kindle — and all screen resolutions must be compatible, too. In the next five years, we'll likely need to design for a number of additional inventions. When will the madness stop? It won't, of course.

In the field of Web design and development, we're quickly getting to the point of being unable to keep up with the endless new resolutions and devices. For many websites, creating a website version for each resolution and new device would be impossible, or at least impractical. Should we just suffer the consequences of losing visitors from one device, for the benefit of gaining visitors from another? Or is there another option?

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. In other words, the website should have the technology to automatically *respond* to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market.

The Concept Of Responsive Web Design

Ethan Marcotte wrote an introductory article about the approach, "Responsive Web Design," for A List Apart. It stems from the notion of responsive architectural design, whereby a room or space automatically adjusts to the number and flow of people within it:

"Recently, an emergent discipline called "responsive architecture" has begun asking how physical spaces can respond to the presence of people passing through them. Through a combination of embedded robotics and tensile materials, architects are experimenting with art installations and wall structures that bend, flex, and expand as crowds approach them. Motion sensors can be paired with climate control systems to adjust a room's temperature and ambient lighting as it fills with people. Companies have already produced "smart glass technology" that can automatically become opaque when a room's occupants reach a certain density threshold, giving them an additional layer of privacy."

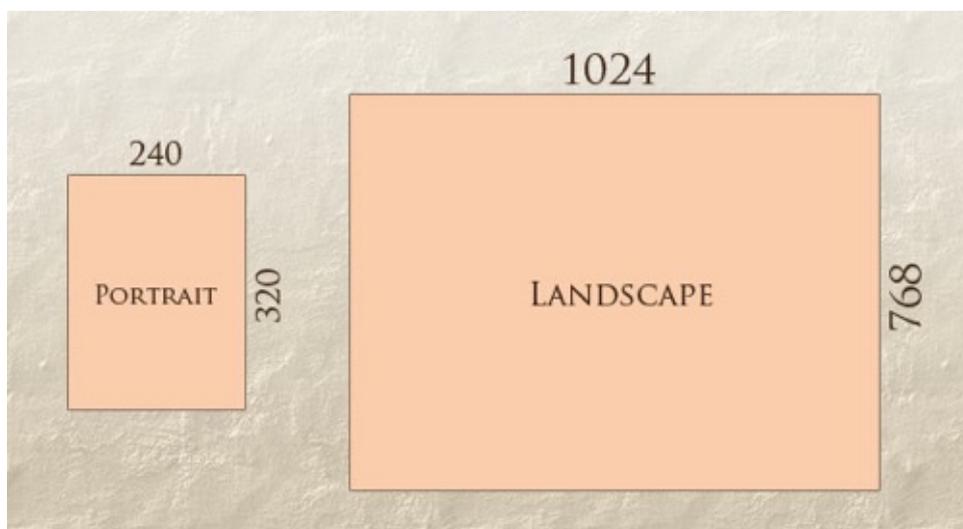
Transplant this discipline onto Web design, and we have a similar yet whole new idea. Why should we create a custom Web design for each group of users; after all, architects don't design a building for each group size and type that passes through it? Like responsive architecture, Web design should automatically adjust. It shouldn't require countless custom-made solutions for each new category of users.

Obviously, we can't use motion sensors and robotics to accomplish this the way a building would. Responsive Web design requires a more abstract way of thinking. However, some ideas are already being practiced: fluid layouts, media queries and scripts that can reformat Web pages and mark-up effortlessly (or *automatically*).

But responsive Web design is not only about adjustable screen resolutions and automatically resizable images, but rather about a whole new way of thinking about design. Let's talk about all of these features, plus additional ideas in the making.

Adjusting Screen Resolution

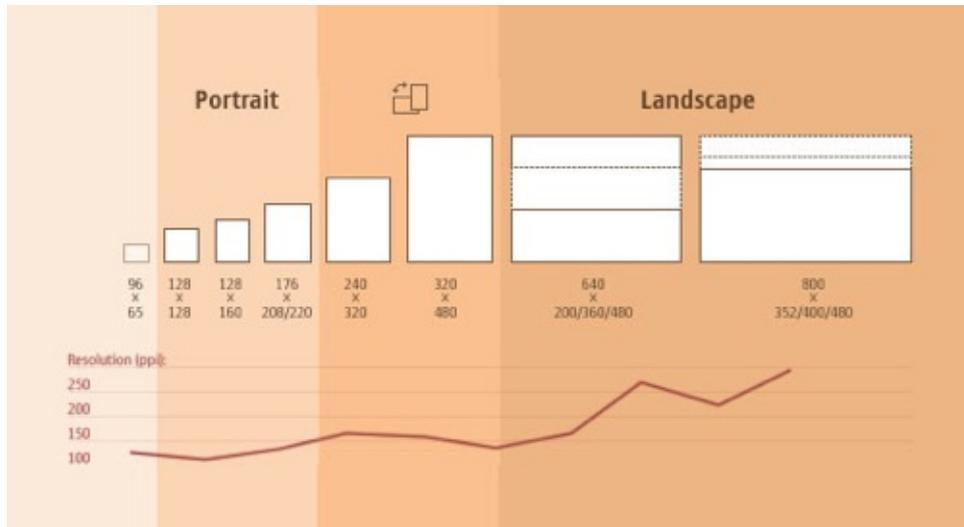
With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim. How is one to design for these situations?



In addition to designing for both landscape and portrait (and enabling those orientations to possibly switch in an instant upon page load), we must consider the hundreds of different screen sizes. Yes, it is possible to group them into major categories, design for each of them, and make each design as flexible as necessary. But that can be overwhelming, and who knows what the usage figures will be in five years? Besides, many users do not maximize their browsers, which itself leaves far too

much room for variety among screen sizes.

Morten Hjerde and a few of his colleagues identified statistics on about 400 devices sold between 2005 and 2008. Below are some of the most common:



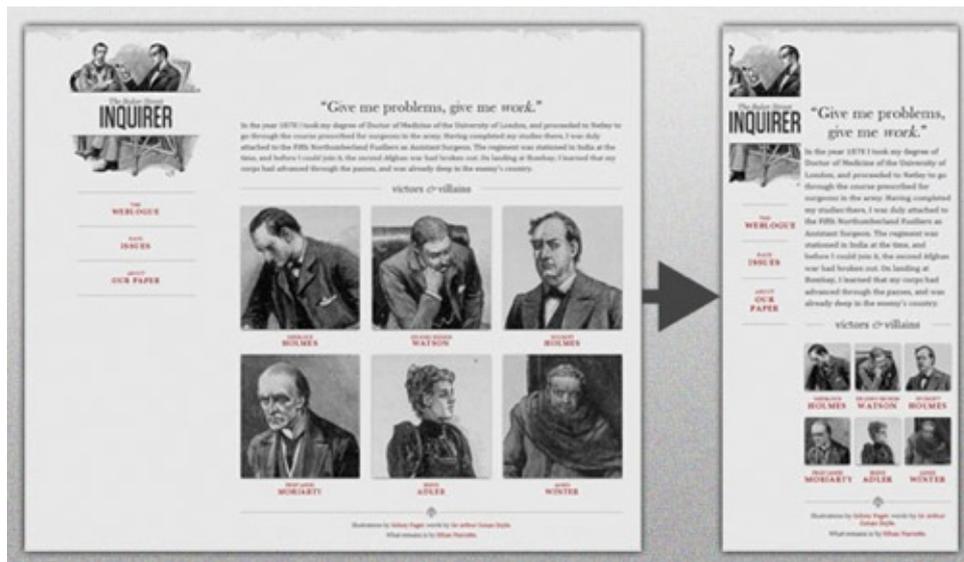
Since then even more devices have come out. It's obvious that we can't keep creating custom solutions for each one. So, how do we deal with the situation?

PART OF THE SOLUTION: FLEXIBLE EVERYTHING

A few years ago, when flexible layouts were almost a "luxury" for websites, the only things that were flexible in a design were the layout columns (structural elements) and the text. Images could easily break layouts, and even flexible structural elements broke a layout's form when pushed enough. Flexible designs weren't really that flexible; they could give or take a few hundred pixels, but they often couldn't adjust from a large computer screen to a netbook.

Now we can make things more flexible. Images can be automatically adjusted, and we have workarounds so that layouts never break (although they may become squished and illegible in the process). While it's not a complete fix, the solution gives us far more options. It's perfect for devices that switch from portrait orientation to landscape in an instant or for when users switch from a large computer screen to an iPad.

In Ethan Marcotte's article, he created a sample Web design that features this better flexible layout:

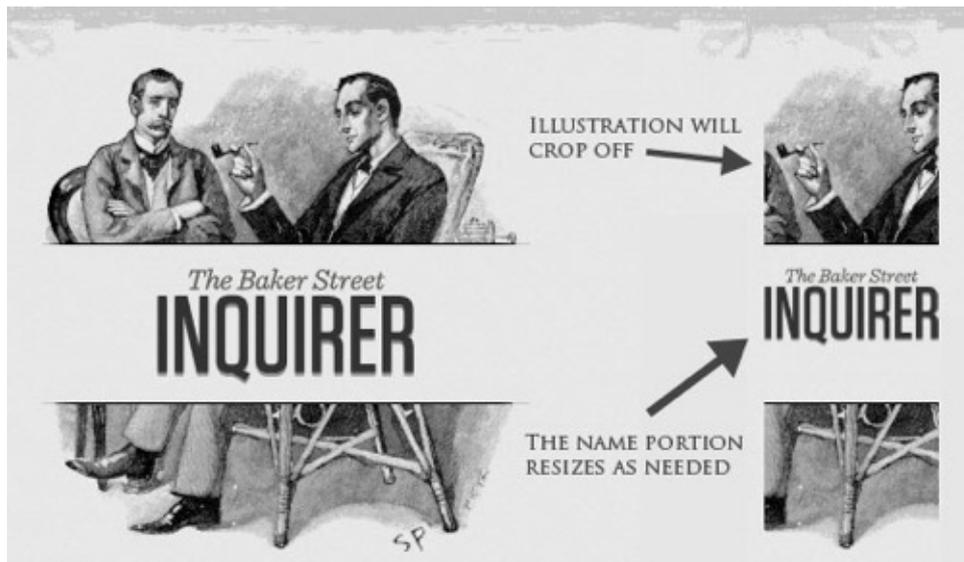


The entire design is a lovely mix of fluid grids, fluid images and smart mark-up where needed. Creating fluid grids is fairly common practice, and there are a number of techniques for creating fluid images:

- [Hiding and Revealing Portions of Images](#)
- [Creating Sliding Composite Images](#)
- [Foreground Images That Scale With the Layout](#)

For more information on creating fluid websites, be sure to look at the book “Flexible Web Design: Creating Liquid and Elastic Layouts with CSS” by Zoe Mickley Gillenwater, and download the sample chapter “Creating Flexible Images.” In addition, Zoe provides the following extensive list of tutorials, resources, inspiration and best practices on creating flexible grids and layouts: “Essential Resources for Creating Liquid and Elastic Layouts.”

While from a technical perspective this is all easily possible, it’s not just about plugging these features in and being done. Look at the logo in this design, for example:



If resized too small, the image would appear to be of low quality, but keeping the name of the website visible and not cropping it off was important. So, the image is divided into two: one (of the illustration) set as a background, to be cropped and to maintain its size, and the other (of the name) resized proportionally.

```
<h1 id="logo"><a href="#"></a></h1>
```

Above, the h1 element holds the illustration as a background, and the image is aligned according to the container’s background (the heading).

This is just one example of the kind of thinking that makes responsive Web design truly effective. But even with smart fixes like this, a layout can become too narrow or short to look right. In the logo example above (although it works), the ideal situation would be to not crop half of the illustration or to keep the logo from being so small that it becomes illegible and “floats” up.

Flexible Images

One major problem that needs to be solved with responsive Web design is working with images. There are a number of techniques to resize images proportionately, and many are easily done. The most popular option, noted in Ethan Marcotte’s article on fluid images but first experimented with by Richard Rutter, is to use CSS’s **max-width** for an easy fix.

As long as no other width-based image styles override this rule, every image will load in its original size, unless the viewing area becomes narrower than the image’s original width. The maximum width of the image is set to 100% of the screen or browser width, so when that 100% becomes narrower, so does the image. Essentially, as Jason Grigsby noted, “The idea behind fluid images is that you deliver images at the maximum size they will be used at. You don’t declare the height and width in your

code, but instead let the browser resize the images as needed while using CSS to guide their relative size”. It’s a great and simple technique to resize images beautifully.

Note that **max-width** is not supported in IE, but a good use of **width: 100%** would solve the problem neatly in an IE-specific style sheet. One more issue is that when an image is resized too small in some older browsers in Windows, the rendering isn’t as clear as it ought to be. There is a JavaScript to fix this issue, though, found in Ethan Marcotte’s article.

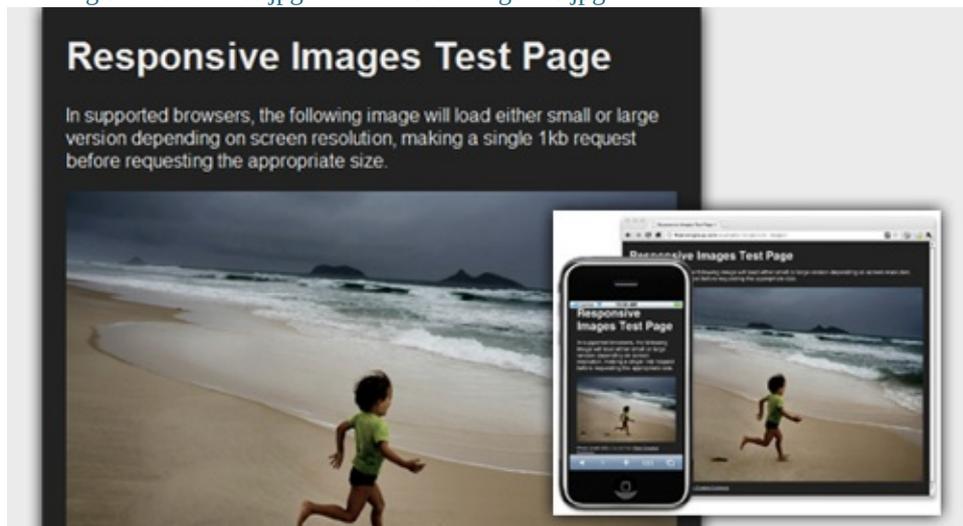
While the above is a great quick fix and good start to responsive images, image resolution and download times should be the primary considerations. While resizing an image for mobile devices can be very simple, if the original image size is meant for large devices, it could significantly slow download times and take up space unnecessarily.

FILAMENT GROUP’S RESPONSIVE IMAGES

This technique, presented by the Filament Group, takes this issue into consideration and not only resizes images proportionately, but shrinks image resolution on smaller devices, so very large images don’t waste space unnecessarily on small screens.

```

```



This technique requires a few files, all of which are available on Github. First, a JavaScript file (*rwd-images.js*), the *.htaccess* file and an image file (*rwd.gif*). Then, we can use just a bit of HTML to reference both the larger and smaller resolution images: first, the small image, with an *.r* prefix to clarify that it should be responsive, and then a reference to the bigger image using **data-fullsrc**.

The **data-fullsrc** is a custom HTML5 attribute, defined in the files linked to above. For any screen that is wider than 480 pixels, the larger-resolution image (*largeRes.jpg*) will load; smaller screens wouldn’t need to load the bigger image, and so the smaller image (*smallRes.jpg*) will load.

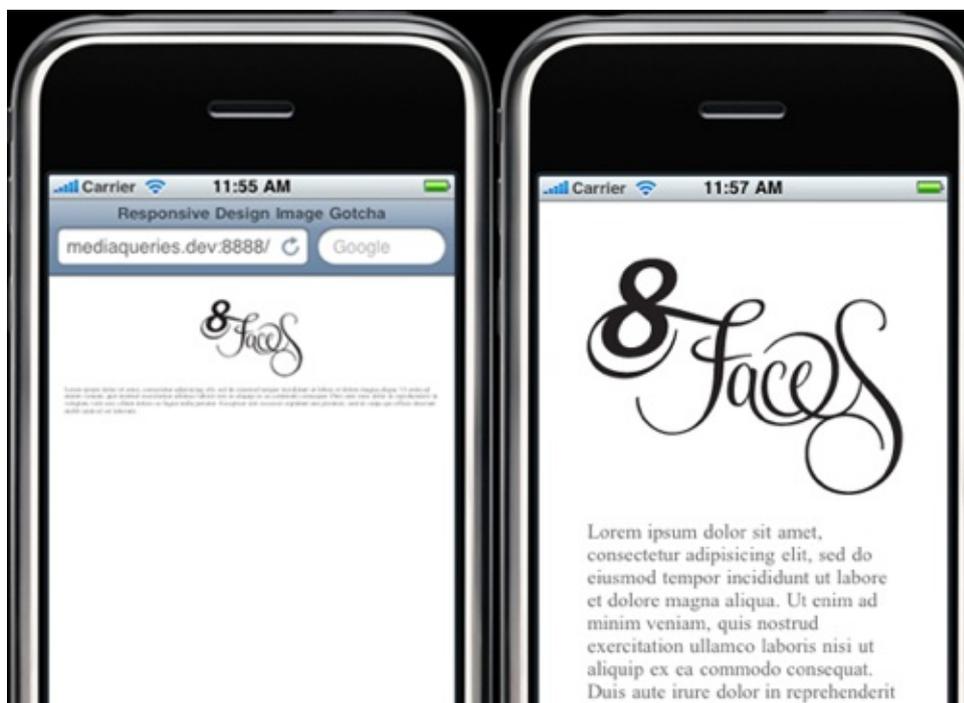
The JavaScript file inserts a base element that allows the page to separate responsive images from others and redirects them as necessary. When the page loads, all files are rewritten to their original forms, and only the large or small images are loaded as necessary. With other techniques, all higher-resolution images would have had to be downloaded, even if the larger versions would never be used. Particularly for websites with a lot of images, this technique can be a great saver of bandwidth and loading time.

This technique is fully supported in modern browsers, such as IE8+, Safari, Chrome and Opera, as well as mobile devices that use these same browsers (iPad, iPhone, etc.). Older browsers and Firefox degrade nicely and still resize as one would expect of a responsive image, except that both resolutions are downloaded together, so the end benefit of saving space with this technique is void.

STOP IPHONE SIMULATOR IMAGE RESIZING

One nice thing about the iPhone and iPod Touch is that Web designs automatically rescale to fit the tiny screen. A full-sized design, unless specified otherwise, would just shrink proportionally for the tiny browser, with no need for scrolling or a mobile version. Then, the user could easily zoom in and out as necessary.

There was, however, one issue this simulator created. When responsive Web design took off, many noticed that images were still changing proportionally with the page even if they were specifically made for (or could otherwise fit) the tiny screen. This in turn scaled down text and other elements.



(Image: Think Vitamin | Website referenced: 8 Faces)

Because this works only with Apple's simulator, we can use an Apple-specific meta tag to fix the problem, placing it *below* the website's **<head>** section. Thanks to Think Vitamin's article on image resizing, we have the meta tag below:

Setting the **initial-scale** to **1** overrides the default to resize images proportionally, while leaving them as is if their width is the same as the device's width (in either portrait or landscape mode). Apple's documentation has a lot more information on the viewport meta tag.

Custom Layout Structure

For extreme size changes, we may want to change the layout altogether, either through a separate style sheet or, more efficiently, through a CSS media query. This does not have to be troublesome; most of the styles can remain the same, while specific style sheets can inherit these styles and move elements around with floats, widths, heights and so on.

For example, we could have one main style sheet (which would also be the default) that would define all of the main structural elements, such as **#wrapper**, **#content**, **#sidebar**, **#nav**, along with colors, backgrounds and typography. Default flexible widths and floats could also be defined.

If a style sheet made the layout too narrow, short, wide or tall, we could then detect that and switch to a new style sheet. This new child style sheet would adopt everything from the default style sheet and then just redefine the layout's structure.

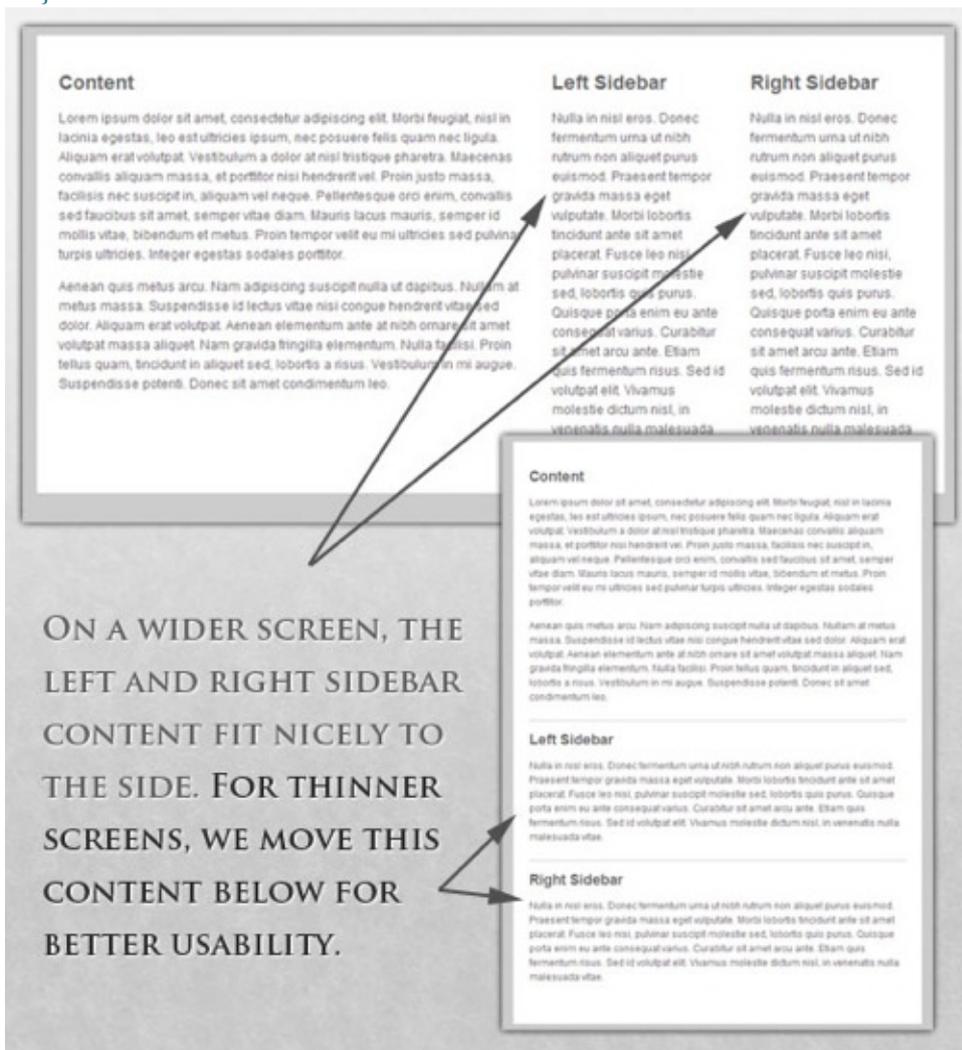
Here is the *style.css* (default) content:

```
/* Default styles that will carry to the child style sheet */
html,body{
  background...
  font...
  color...
}
h1,h2,h3{}
p,blockquote,pre,code,ol,ul{}
/* Structural elements */
#wrapper{
  width: 80%;
  margin: 0 auto;
  background: #fff;
  padding: 20px;
}
#content{
  width: 54%;
  float: left;
  margin-right: 3%;
}
#sidebar-left{
  width: 20%;
  float: left;
  margin-right: 3%;
```

```

}
#sidebar-right{
  width: 20%;
  float: left;
}
}
Here is the mobile.css (child) content:
#wrapper{
  width: 90%;
}
}
#content{
  width: 100%;
}
}
#sidebar-left{
  width: 100%;
  clear: both;
  /* Additional styling for our new layout */
  border-top: 1px solid #ccc;
  margin-top: 20px;
}
}
#sidebar-right{
  width: 100%;
  clear: both;
  /* Additional styling for our new layout */
  border-top: 1px solid #ccc;
  margin-top: 20px;
}
}

```



MEDIA QUERIES

CSS3 supports all of the same media types as CSS 2.1, such as **screen**, **print** and **handheld**, but has added dozens of new media features, including **max-width**, **device-width**, **orientation** and **color**. New devices made after the release of CSS3 (such as the iPad and Android devices) will definitely support media features. So, calling a media query using CSS3 features to target these devices would work just fine, and it will be ignored if accessed by an older computer browser that does not support CSS3.

In Ethan Marcotte’s article, we see an example of a media query in action:

```
<link rel="stylesheet" type="text/css"
      media="screen and (max-device-width: 480px)"
      href="shetland.css" />
```

This media query is fairly self-explanatory: if the browser displays this page on a screen (rather than print, etc.), and if the width of the screen (not necessarily the viewport) is 480 pixels or less, then load *shetland.css*.

New CSS3 features also include **orientation** (portrait vs. landscape), **device-width**, **min-device-width** and more. Look at “The Orientation Media Query” for more information on setting and restricting widths based on these media query features.

One can create multiple style sheets, as well as basic layout alterations defined to fit ranges of widths — even for landscape vs. portrait orientations. Be sure to look at the section of Ethan Marcotte’s article entitled “Meet the media query” for more examples and a more thorough explanation.

Multiple media queries can also be dropped right into a single style sheet, which is the most efficient option when used:

```
/* Smartphones (portrait and landscape) ----- */
@media only screen
and (min-device-width : 320px)
and (max-device-width : 480px) {
/* Styles */
}
/* Smartphones (landscape) ----- */
@media only screen
and (min-width : 321px) {
/* Styles */
}
/* Smartphones (portrait) ----- */
@media only screen
and (max-width : 320px) {
/* Styles */
}
```

The code above is from a free template for multiple media queries between popular devices by Andy Clark. See the differences between this approach and including different style sheet files in the mark-up as shown in the post “Hardboiled CSS3 Media Queries.”

CSS3 MEDIA QUERIES

Above are a few examples of how media queries, both from CSS 2.1 and CSS3 could work. Let's now look at some specific how-to's for using CSS3 media queries to create responsive Web designs. Many of these uses are relevant today, and all will definitely be usable in the near future.

The `min-width` and `max-width` properties do exactly what they suggest. The **min-width** property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored. The **max-width** property does just the opposite. Anything above the maximum browser or screen width specified would not apply to the respective media query.

Note in the examples below that we're using the syntax for media queries that could be used all in one style sheet. As mentioned above, the most efficient way to use media queries is to place them all in one CSS style sheet, with the rest of the styles for the website. This way, multiple requests don't have to be made for multiple style sheets.

```
@media screen and (min-width: 600px) {  
  .hereIsMyClass {  
    width: 30%;  
    float: right;  
  }  
}
```

The class specified in the media query above (**hereIsMyClass**) will work only if the browser or screen width is above 600 pixels. In other words, this media query will run only if the minimum width is 600 pixels (therefore, 600 pixels or wider).

```
@media screen and (max-width: 600px) {  
  .aClassforSmallScreens {  
    clear: both;  
    font-size: 1.3em;  
  }  
}
```

Now, with the use of **max-width**, this media query will apply only to browser or screen widths with a maximum width of 600 pixels or narrower.

While the above **min-width** and **max-width** can apply to either screen size or browser width, sometimes we'd like a media query that is relevant to device width specifically. This means that even if a browser or other viewing area is minimized to something smaller, the media query would still apply to the size of the actual device. The `min-device-width` and `max-device-width` media query properties are great for targeting certain devices with set dimensions, without applying the same styles to other screen sizes in a browser that mimics the device's size.

```
@media screen and (max-device-width: 480px) {  
  .classForiPhoneDisplay {  
    font-size: 1.2em;  
  }  
}
```

```

    }
  }
  @media screen and (min-device-width: 768px) {
    .minimumiPadWidth {
      clear: both;
      margin-bottom: 2px solid #ccc;
    }
  }
}

```

There are also other tricks with media queries to target specific devices. Thomas Maier has written two short snippets and explanations for targeting the iPhone and iPad only:

- [CSS for iPhone 4 \(Retina display\)](#)
- [How To: CSS for the iPad](#)

For the iPad specifically, there is also a media query property called orientation. The value can be either landscape (horizontal orientation) or portrait (vertical orientation).

```

@media screen and (orientation: landscape) {
  .iPadLandscape {
    width: 30%;
    float: right;
  }
}
@media screen and (orientation: portrait) {
  .iPadPortrait {
    clear: both;
  }
}

```

Unfortunately, this property works only on the iPad. When determining the orientation for the iPhone and other devices, the use of **max-device-width** and **min-device-width** should do the trick.

There are also many media queries that make sense when combined. For example, the **min-width** and **max-width** media queries are combined all the time to set a style specific to a certain range.

```

@media screen and (min-width: 800px) and (max-width: 1200px) {
  .classForaMediumScreen {
    background: #cc0000;
    width: 30%;
    float: right;
  }
}

```

The above code in this media query applies only to screen and browser widths between 800 and 1200 pixels. A good use of this technique is to show certain content or entire sidebars in a layout depending on how much horizontal space is available.

Some designers would also prefer to link to a separate style sheet for certain media queries, which is perfectly fine if the organizational benefits outweigh the efficiency lost. For devices that do not switch orientation or for screens whose browser width cannot be changed manually, using a separate style sheet should be fine.

You might want, for example, to place media queries all in one style sheet (as above) for devices like the iPad. Because such a device can switch from portrait to landscape in an instant, if these two media queries were placed in separate style sheets, the website would have to call each style sheet file every time the user switched orientations. Placing a media query for both the horizontal and vertical orientations of the iPad in the same style sheet file would be far more efficient.

Another example is a flexible design meant for a standard computer screen with a resizable browser. If the browser can be manually resized, placing all variable media queries in one style sheet would be best.

Nevertheless, organization can be key, and a designer may wish to define media queries in a standard HTML link tag:

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css" />
<link rel="stylesheet" media="screen and (min-width: 600px)" href="large.css" />
<link rel="stylesheet" media="print" href="print.css" />
```

JAVASCRIPT

Another method that can be used is JavaScript, especially as a back-up to devices that don't support all of the CSS3 media query options. Fortunately, there is already a pre-made JavaScript library that makes older browsers (IE 5+, Firefox 1+, Safari 2) support CSS3 media queries. If you're already using these queries, just grab a copy of the library, and include it in the mark-up: *css3-mediaqueries.js*.

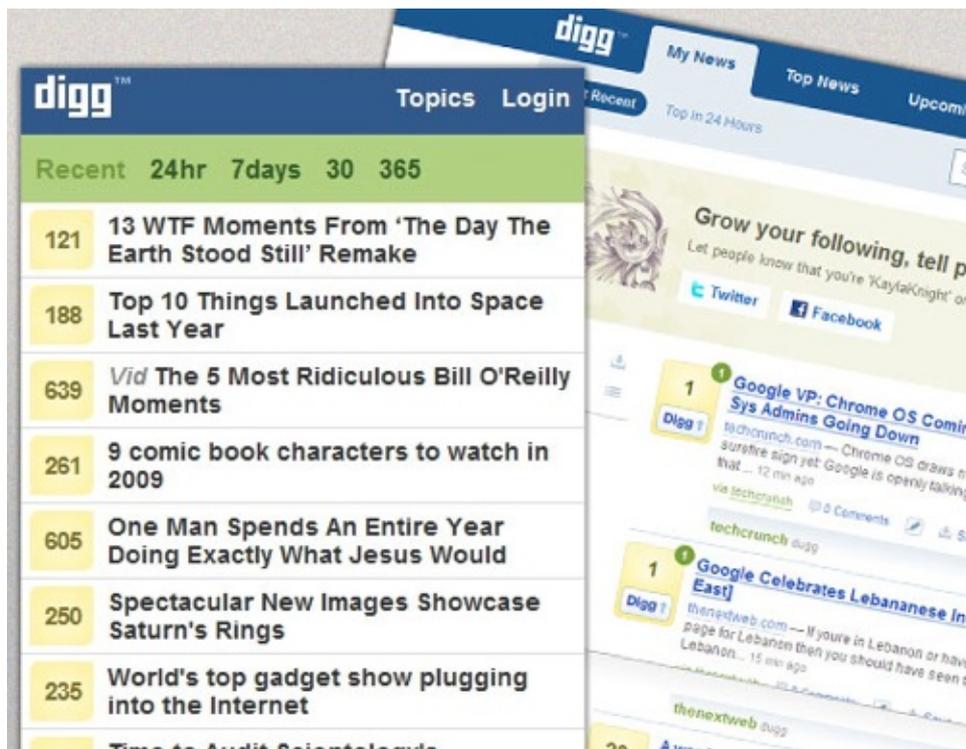
In addition, below is a sample jQuery snippet that detects browser width and changes the style sheet accordingly — if one prefers a more hands-on approach:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js">
</script>
<script type="text/javascript">
    $(document).ready(function(){
        $(window).bind("resize", resizeWindow);
        function resizeWindow(e){
            var newWindowWidth = $(window).width();
            // If width is below 600px, switch to the mobile stylesheet
            if(newWindowWidth < 600){
                $("link[rel=stylesheet]").attr({href :
"mobile.css"});
            } // Else if width is above 600px, switch to the large
stylesheet
            else if(newWindowWidth > 600){
                $("link[rel=stylesheet]").attr({href : "style.css"});
            }
        }
    });
</script>
```

There are many solutions for pairing up JavaScript with CSS media queries. Remember that media queries are not an absolute answer, but rather are fantastic options for responsive Web design when it comes to pure CSS-based solutions. With the addition of JavaScript, we can accommodate far more variations. For detailed information on using JavaScript to mimic or work with media queries, look at “Combining Media Queries and JavaScript.”

Showing or Hiding Content

It is possible to shrink things proportionally and rearrange elements as necessary to make everything fit (reasonably well) as a screen gets smaller. It's great that that's possible, but making every piece of content from a large screen available on a smaller screen or mobile device isn't always the best answer. We have best practices for mobile environments: simpler navigation, more focused content, lists or rows instead of multiple columns.



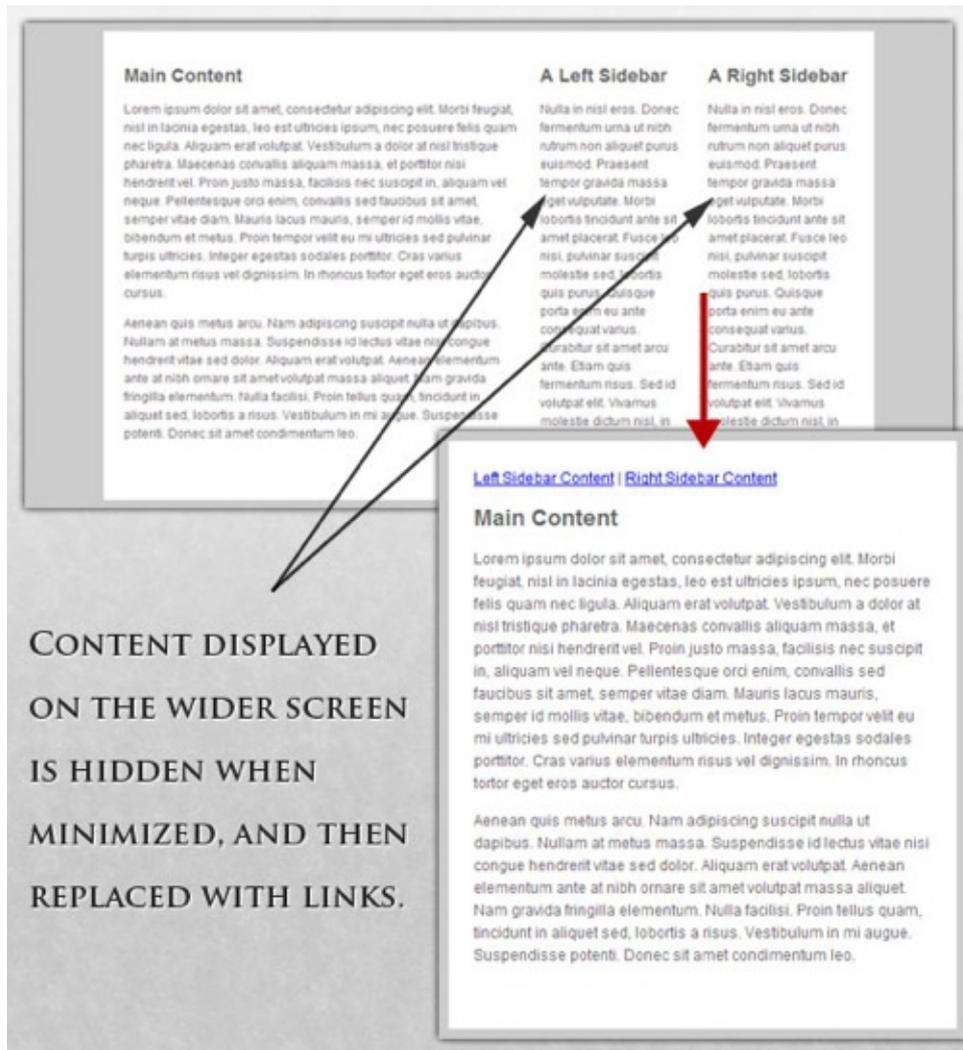
Responsive Web design shouldn't be just about how to create a flexible layout on a wide range of platforms and screen sizes. It should also be about the user being able to pick and choose content. Fortunately, CSS has been allowing us to show and hide content with ease for years!

```
display: none;
```

Either declare **display: none** for the HTML block element that needs to be hidden in a specific style sheet or detect the browser width and do it through JavaScript. In addition to hiding content on smaller screens, we can also hide content in our default style sheet (for bigger screens) that should be available only in mobile versions or on smaller devices. For example, as we hide major pieces of content, we could replace them with navigation to that content, or with a different navigation structure altogether.

Note that we haven't used **visibility: hidden** here; this just hides the content (although it is still there), whereas the `display` property gets rid of it altogether. For smaller devices, there is no need to keep the mark-up on the page — it just takes up resources

and might even cause unnecessary scrolling or break the layout.



Here is our mark-up:

```
<p class="sidebar-nav"><a href="#">Left Sidebar Content</a> | <a href="#">Right Sidebar Content</a>
</p>
<div id="content">
  <h2>Main Content</h2>
</div>
<div id="sidebar-left">
  <h2>A Left Sidebar</h2>
</div>
<div id="sidebar-right">
  <h2>A Right Sidebar</h2>
</div>
```

In our default style sheet below, we have hidden the links to the sidebar content. Because our screen is large enough, we can display this content on page load.

Here is the **style.css** (default) content:

```
#content{
```

```

        width: 54%;
        float: left;
        margin-right: 3%;
    }

    #sidebar-left{
        width: 20%;
        float: left;
        margin-right: 3%;
    }

    #sidebar-right{
        width: 20%;
        float: left;
    }
    .sidebar-nav{display: none;}

```

Now, we hide the two sidebars (below) and show the links to these pieces of content. As an alternative, the links could call to JavaScript to just cancel out the **display: none** when clicked, and the sidebars could be realigned in the CSS to float below the content (or in another reasonable way).

Here is the **mobile.css** (simpler) content:

```

#content{
    width: 100%;
}
#sidebar-left{
    display: none;
}
#sidebar-right{
    display: none;
}
.sidebar-nav{display: inline;}

```

With the ability to easily show and hide content, rearrange layout elements and automatically resize images, form elements and more, a design can be transformed to fit a huge variety of screen sizes and device types. As the screen gets smaller, rearrange elements to fit mobile guidelines; for example, use a script or alternate style sheet to increase white space or to replace image navigation sources on mobile devices for better usability (icons would be more beneficial on smaller screens).

TOUCHSCREENS VS. CURSORS

Touchscreens are becoming increasingly popular. Assuming that smaller devices are more likely to be given touchscreen functionality is easy, but don't be so quick. Right now touchscreens are mainly on smaller devices, but many laptops and desktops on the market also have touchscreen capability. For example, the HP Touchsmart tm2t is a basic touchscreen laptop with traditional keyboard and mouse that can transform into a tablet.