

# Optimizing Sites for Mobile Devices

James Williamson  
Senior Author  
lynda.com

<b>OPTIMIZING SITES FOR MOBILE DEVICES .....</b>	<b>1</b>
<b>Exercise 1: Developing a Mobile Strategy .....</b>	<b>2</b>
<b>Exercise 2: Defining Media Queries.....</b>	<b>5</b>
<b>Exercise 3: Creating Fluid Layouts.....</b>	<b>7</b>
<b>Exercise 4: Creating fluid regions .....</b>	<b>12</b>
<b>Exercise 5: Optimizing Images for Mobile Devices.....</b>	<b>15</b>
<b>Exercise 6: Designing for the Mobile Context.....</b>	<b>22</b>
<b>Exercise 7: Managing Resources.....</b>	<b>28</b>
<b>Exercise 9: Enhancing Mobile Experiences.....</b>	<b>35</b>

## Exercise 1: Developing a Mobile Strategy

Often you don't realize fundamental shifts in an industry have happened until much later, when you can look back and reflect on the significance of the changes that were made. In web design, we've seen several of those fundamental shifts –the rise of CSS, the adoption of Content Management Systems, and the emergence of web applications just to name a few– and have had to adapt our design skills and strategies accordingly. I believe that we're in the middle of another fundamental shift, and one that many web designers aren't properly addressing yet.

Over the last four years, how people connect to and consume online content has changed dramatically. The rise of smartphones, tablets, and other internet capable devices have given consumers a variety of new ways to experience and interact with your content. While there are countless studies and statistics that will illuminate the numbers I'm talking about, all you really need to do is think about is how you, and the people you know, use the Internet. My guess is that a mobile device is a large part of your browsing experience.

Despite this, many designers either ignore mobile devices, or design for mobile only if the client requests it. A recent, non-scientific survey on a popular CSS forum asked designers how they dealt with designing for mobile devices. Only 15% said they include a mobile-specific stylesheet, 9% said they redirect users to a mobile site, another 9% said they use a 3<sup>rd</sup> party re-direct, and another 9% said that they didn't think it was necessary. Perhaps even worse, 44% said they design for mobile only if the client requested it, while the remaining 15% said they had never really considered it.

While these numbers might not be indicative of the web community at large, clearly there is a disconnect between how people are using the web and the current state of web design. By failing to address mobile devices, web designers are creating poor user experiences for the people consuming their sites, and doing the client a disservice by not properly designing for how people will use their sites. While the argument about whether or not designing for mobile should be included in any site design agreement is the topic for another discussion, the point I'm trying to make is that as designers, we can no longer ignore mobile devices, or act as though they are a special-needs case for specific sites. Mobile devices are here for the long-run, and we'd better get used to it.

### **Current options for mobile devices**

Currently, there are four main options when designing for mobile devices. You can develop a separate mobile application, develop a separate site specifically for mobile devices and redirect mobile users to the site, optimize your site to work properly across multiple devices, or ignore mobile devices altogether. The first and second options are often the best choices, depending upon the type of content being presented, the goals of the client, and the context in which the content should be presented to the device. In many cases however, the first two choices may not be the best option, or within the capabilities of the client or designer. In those cases,

designers are left with the choice of attempting to tailor the design to work optimally across multiple devices, or simply ignoring mobile devices and hoping the result will be sufficient. I feel strongly that regardless of the budget or goals of a project that ignoring mobile devices is a poor choice. In fact, I argue that by focusing on mobile devices as an integral part of your web design strategy, your sites will be more focused, work better, and meet the goals of you clients in ways that ignoring them can never do.

### **What is your mobile strategy?**

To develop for mobile devices, you need to have a coherent strategy. A recent blog post by Luke Wroblewski [<http://www.lukew.com/ff/entry.asp?933>] put forward the idea of “mobile first” as a way of looking at how designing for the mobile context can change your design. Luke puts forward two important benefits of considering mobile devices first when designing for the web. First, the focus that designing for mobile brings to your projects in regards to the importance of content and how it is presented, and the new capabilities that mobile brings us in terms of how you can present content to your users.

While not everyone will want to begin designing for mobile devices first before designing for desktop, I recommend at least designing for mobile in tandem with the desktop. Too often people see mobile design as something to do *after* they finish the real design for the desktop. This approach costs designers a tremendous amount of time, forces important design and architecture decisions to be made much later in the process, and reduces mobile to a 2<sup>nd</sup> class status that rarely results in the mobile and desktop sites complementing each other. Often the best result from this approach is a mobile site that is merely a watered down version of the desktop site. To me, it goes without saying that this is hardly ideal for such an important segment of your audience.

### **Developing for multiple screens**

While developing for the mobile context is not the same as developing for the desktop, you can focus the design for both platforms by designing for them simultaneously. In this exercise, we’ll take a look at a sample site mock-up, and discuss the decisions that will drive the approach for each device.

### **Objectives**

This exercise will focus on

- Options for designing for mobile devices
- What “mobile first” means in terms of developing a design strategy
- Understanding mobile considerations
- Planning for mobile devices

## Steps

Launch Adobe Illustrator and open the *desolve\_mockup.ai* in the Exercise\_01 folder

1. Follow along with the instructor as he describes the design process and considerations that need to be made for multiple devices.
2. Focus on the following questions:
  - a. How do the needs of content change when screen sizes change?
  - b. How do the different presentations change how the page content should be structured?
  - c. How does designing for multiple screen sizes at once assist you in planning document styles?
  - d. What types of alternate approaches to presenting and structuring content could you do that are specific to each medium?
  - e. Are designing for screen sizes and designing for mobile devices the same thing?
  - f. When is these particular approach -designing flexible layouts within screen size ranges- appropriate, and when is it not?

## Exercise 2: Defining Media Queries

Media queries allow you to determine which styles are applied to pages based on specific media properties such as screen width, color, or resolution. If you're familiar with using the media attribute to apply styles for screen devices or print devices, media queries will feel quite natural, as they are an extension of existing media capabilities. For our desolve.org site, we'll use screen-width to determine which sets of styles to load. While screen-width is **not** an indication of which device is being used, it is a fairly robust way of ensuring that the correct styles are supplied to proper devices. We'll also use the *min*- and *max*- prefixes to provide a range of screen sizes for each set of styles. This allows us to account for small variations in screen sizes for different types of mobile devices.

### Objectives

This exercise will focus on

- Understanding what media queries are
- Structuring external CSS files for device-specific styles
- Using Dreamweaver to establish media queries for our pages

### Steps

#### Creating external style sheets for mobile devices

1. Open the index.htm page from the **Exercise\_02** folder.
2. Note that the stylesheet **main.css** is already attached to it.
3. To set up media queries, go to the menu and choose **Modify > Media Queries**, or click the pulldown menu on the **Multiscreen Preview** button and choose Media Queries.

#### Adding Media Queries

The Media Queries dialog box can be a bit overwhelming at first glance. The initial set of choices allow you to define whether you are setting up media queries for the entire site, or local to the page you're working on. The second section allows you to define your individual media queries. Here you can add as many media queries as you wish to your page or site, and then modify the queries for your specific needs.

4. Choose the **This document** option in the **Write media queries to** section.
5. Uncheck the **Force devices to report actual width** checkbox. Although we will be doing this for our site, we'll specify this in more detail later.
6. Click the Default Presets button in the lower right-hand corner of the dialog box. This will populate a list of media queries in the dialog box and provide you

with a solid set of media queries, or a good starting point for a custom list. You also have the option to add each query manually if you wish.

7. Select the first media query and change its description to **mobile**.
8. Change its **max-width** value to **480**.
9. For CSS file, choose the **Create New File** option and browse to the **Exercise\_02/\_css** directory.
10. **Save** to file as **mobile.css**.
11. Select the second media query and change its description to **tablet**.
12. Change its **min-width** value to **481**, leave its **max-width** value at **768**.
13. For CSS file, choose the **Create New File** option and browse to the **Exercise\_02/\_css** directory.
14. **Save** to file as **tablet.css**.
15. Select the third media query and change its description to **desktop**.
16. Leave its **min-width** value at **769**.
17. For CSS file, choose the **Create New File** option and browse to the **Exercise\_02/\_css** directory.
18. **Save** to file as **desktop.css**.
19. Click OK to close the media query dialog. Review the links created in the head of the document, and the empty CSS files now listed in related documents.

### Using media queries

We now have four separate CSS files. One, `main.css` is limited to only screen and projection devices. The others, `mobile.css`, `tablet.css`, and `desktop.css` will only be used if the reported screen sizes fall into its specific ranges. We may now author both our global styles, and screen specific styles within these documents.

## Exercise 3: Creating Fluid Layouts

One of the biggest issues that designers face when designing for mobile devices is the wide variety of screen sizes among the various mobile devices. Locking your designs at a specific pixel size not only limits you to a specific platform, it doesn't take into account future devices and screen sizes. Fluid layouts adapt to the size of the screen, making them an ideal layout solution for situations where the exact size of the device isn't known before-hand. Where fluid layouts can fail, however, is when screen sizes fluctuate too widely. Multicolumn layouts often fail, with content overlapping or dropping when the column widths become too small to support the content within. To keep this from happening, we'll use three separate fluid layouts, one for desktop, one for tablets, and one for mobile devices. We'll also set minimum and maximum widths that are relevant to the targeted devices. That way, the layouts will be fluid, but only within certain ranges, which allow for slight differences in device screen size.

### Objectives

This exercise will focus on

- Layout strategies for multiple devices
- Creating fluid layouts
- Testing the effect of screen resizing on multiple layouts
- Using Dreamweaver's Multiscreen Preview

### Steps

#### Exploring page structure

1. Open the **index.htm** file in the **Exercise\_03** folder.
2. Switch to **Code View** and note the page structure. All content other than the basic structural elements have been removed.
3. Click on the **main.css** file in the related files list. Note the presence of a limited CSS reset, and a rule to ensure block level display for new HTML5 elements.

#### Setting global values

In some cases, the layout styles for certain elements will remain consistent from layout to layout. Rather than repeating those rules in each stylesheet, we'll declare them in the **main.css** file so they are shared by all layouts.

4. If you prefer to hand-code, I'll provide you with the complete rule at the beginning of each new rule. If you prefer to create rules visually, follow the step-by-step instructions given for the first rule, and then repeat them for each successive rule, using the provided values.

5. We'll start by formatting the header. The complete rule:

```
header {background: rgb(100, 98, 102); margin-bottom: 1.5em;
height: 175px; }
```

6. In the **CSS Styles** panel, click the **new rule** icon.
7. Choose **tag** for **selector** type, choose **header** as the tag, and define the rule in the **main.css** file.
8. Click on the **background** category. Click on the color chip for background color, click the fly-out menu in the top right corner, and choose **rgb()** for color format.
9. Click the color wheel icon to use the **system color picker**. Change to **RGB sliders** and mix the color **R: 100, G: 98, and B: 102**. Since we'll use this color again, drag it down to your saved colors so that you won't have to re-mix this color.
10. Click on the **box model** category and set **margin-bottom to 1.5em**, and **height to 175px**. Height isn't really needed, but will allow us to visualize our structural elements.
11. Click OK to create the rule.
12. Create the **#mainNav** rule:

```
#mainNav { height: 100px; margin-bottom: 1.5em; background:
rgb(102, 127, 106) }
```

13. Using the same steps as above, in the new rule dialog box, go to the **background category** and set the **background to rgb(102, 127, 106)**.
14. In the **Box category** set the **margin-bottom to 1.5em** and **height to 100px**.
15. For global styles, we'll simply set backgrounds for each of the remaining main structural elements. Create rules for the article, aside, and page footer elements:

```
article { background:  rgb(123, 121, 143); }
aside {background: rgb(237, 228, 214); }
#pageFooter {background: rgb(216, 213, 188); }
```

16. Give them the following backgrounds:

- a. article: rgb(123,121,143)
- b. aside: rgb(237,228,214)
- c. #pageFooter: rgb(216,213,188)

17. Save the file

### Creating the desktop layout

Now we'll create the layout for the main structural elements for each of the three devices. We'll start with our desktop styles, which should be fluid within a range of 1280 – 1024 pixels. In the interest of saving time, the sectional styles for the mobile and tablet layouts are already complete. Be sure to compare the styles you'll create here with the styles in the other CSS files to understand the differences in the layouts.

18. Make sure you're working with the **desktop.css** file. If you are hand-coding, be sure to select desktop.css from related files. If you are using the CSS Styles panel, make sure you select the desktop.css file from the list of styles in the all view before creating the following styles.

19. Create the following rule for the body tag:

```
body { width: 90%; margin: 0 auto; min-width: 1024px; max-width: 1280px; }
```

20. In the new rule dialog box, choose **tag** for type, select the **body** tag, and set the following rules.

- a. **width: 90%**
- b. **margin: 0** for **top** and **bottom**, **auto** for **left** and **right**

21. Click OK to create the rule and then add the **min-width: 1024px** and **max-width: 1280px** either by hand to the code or through the properties pane of the CSS Styles panels as these are not supported in the new rule dialog.

22. Often you will want sections to have properties that vary from one layout to another. We'll start that process by varying the height in the desktop styles. Create the following rule:

```
header { height: 165px; }
```

We'll create our two-column layout by floating the article to the left, and the aside to the right. Since we want our layout to be fluid, we'll use percentages for the widths and margins.

23. In **desktop.css**, create the following rule for the **article** tag.

```
article { float: left; width: 70.1%; padding-left: 3.9%; height: 300px; }
```

24. In the new rule dialog box, select **tag** from **selector type**, and choose **article** from the list of tags.

25. In the CSS Rule Definition dialog, set the following properties in the **box** category:

- a. float: left
- b. width: 70.1%
- c. padding-left: 3.9%
- d. height: 300px;

26. Next, create the following rule for the **aside** tag.

```
aside { float: right; width: 19.1%; padding-right: 3.9%; height: 300px; text-align: right; }
```

27. Using the steps from the previous selector, choose the **aside** tag for the selector, and set the following properties in the **box category**:

- a. float: right;
- b. width: 19.1%
- c. padding-right: 3.9%

28. In the **block** category set the **text-align** property to **right**.

29. Finally, set the following rule for the #pageFooter element:

```
#pageFooter { clear: both; height: 150px; }
```

30. For the footer, you'll need to choose **ID** from **selector type** and type the **#pageFooter** ID into the selector name field.

31. In the box category, set the following properties

- a. clear: both
- b. height: 150px

32. Save and test your file, resizing the browser to observe the fluid nature of the layout, the minimum and maximum limits of it, and how layout changes are triggered by screen size.

### Using Dreamweaver's Multiscreen Preview

One of my favorite recent additions to Dreamweaver has been the Multiscreen Preview panel. This panel allows you to preview how your page will render using Dreamweaver's internal WebKit rendering engine in three different viewport sizes. The viewport sizes are customizable, and give you the same browsing capabilities that Live View does. We'll experiment with Multiscreen Preview, and change its settings to match our project.

33. In **Design View**, click on the **Multiscreen Preview** icon found in the **document toolbar**.
34. In the upper right-hand corner, click the Viewport Sizes icon. Note that you can also set up media queries from this view.
35. For **phone**, use **320 x 300**. For **tablet** use **768 x 300** and for **desktop** use **1280**. If your screen resolution won't support the heights, modify them to fit the viewing area.
36. Observe how you can **scroll** through the viewports independently of each other, and how the Multiscreen Preview panel can be resized.
37. **Resize** the panel until it covers the majority of the screen.
38. **Double-click** the **panel tab** to collapse it, then double-click it again to bring it back. **Click** the **double-arrows** in the upper right hand corner to collapse the panel to an icon, and then click them again to bring the panel back. Both of these options provide you with a way to have the panel constantly on hand, without cluttering the workspace.

Now, I know that right now our site doesn't look like much, but truthfully a very important step has been taken. We now have our structural areas defined, and styled in such a way so that the page layout responds to the screen size, and allows for slight variations in size for the different target devices. This allows us to have a two-column layout in situations where we have more space, and switch to a single-column layout for smaller devices.

## Exercise 4: Creating fluid regions

Once you've perfected the layout for your main content regions, you'll need to turn your attention towards making sure the content within those sections maintains the same type of flexibility you've given your overall page. This can be especially problematic when dealing with images. Images are, by their nature, fixed in size, and can ruin fluid layouts when content is resized or columns shrink. In this exercise we'll concentrate on controlling the layout of a sample gallery, and in creating fluid images that will scale to the appropriate size for the layout.

### Objectives

This exercise will focus on

- Styling fluid columns
- Creating fluid images
- Using CSS3 multiple columns

### Steps

#### Structuring gallery content

For our desktop layout, we want our text to display in two columns to take advantage of the larger screen real estate and the photos to display in two columns just below the body copy. We'll format our columns by using CSS3's multiple column properties. Before we can properly style our regions, we'll need to add a bit more structure to our gallery.

1. Open the **philadelphia.htm** in the **Exercise\_04** folder.
2. Switch to **code view**, and find the body copy just below the heading 1 (around line **36**)
3. **Highlight** the 5 paragraphs between the gallery heading and the gallery images.
4. Click the **tag editor** icon in the **coding toolbar**, or use the **CMD + T** keyboard shortcut to bring up the tag editor shortcut.
5. In the hinting window that appears, enter **<div class="intro">** to wrap the paragraphs in a div tag with the class of intro. Use code hinting to speed the process of entering the code.
6. Repeat the process with the images. **Highlight all of the images** below the text and wrap them in a div tag with the class of "photos" (**<div class="photos">**)

### Styling multiple columns

7. Switch to **Design** view, in the **CSS Styles panel**, make sure you're in the **all view** and scroll through the styles until you find the **#gallery h1** rule. Click to highlight this rule.
8. Click the **new rule icon** and choose class from selector type. Give the class the name **".intro"**.
9. In the **box** category, give **.intro** a **margin-bottom of 1em**. Click OK to close the dialog box.
10. Make sure the new **.intro** rule is highlighted in the CSS Styles panel.
11. Click the **add property** link in the middle properties pane.
12. Type in **-moz-column-count** in the property section and give it a value of **2**.
13. Using the same steps, add the following properties to the **.intro** rule:
  - a. **-webkit-column-count: 2**
  - b. **column-count: 2**
  - c. **-moz-column-gap: 1em**
  - d. **-webkit-column-gap: 1em**
  - e. **column-gap: 1em**
14. Switch to Live View and preview the changes to your intro text.
15. Often it is faster to work in Code View. Switch to **Code View** and locate the **.intro** rule.
16. **Copy** the rule and **paste** it below the existing **.intro** rule.
17. Change the selector name to **.photos** and change the **margin-bottom to 2em**.
18. **Save** the page and **preview** it in a browser. Resize the page and note how the size of the images is breaking the layout when resized below 1280.

### Making images fluid

19. Return to the **philadelphia.htm** file.
20. Either in Code View, or by using the Properties Inspector in Design View, remove the width and height for all gallery images.
21. Create a new CSS rule with a **selector** of **.photos img** to target all the images in the photos section.

22. Set the **width** to **100%** and the **margin-bottom** property to **.5em**.
23. **Save** the file and preview in a browser. Resize the window and note how the images now resize based on the size of the column. The 100% width tells the image to fit to the width of the column. Height is adjusted automatically by the browser, and the bottom margin provides vertical separation for the images.

## Exercise 5: Optimizing Images for Mobile Devices

One of the biggest challenges designers face when adapting designs to multiple devices is how to handle images from one device context to another. Typically, desktop oriented sites can make liberal use of images, and connection speeds make the size of images less of a concern than it is for mobile devices. In addition, the smaller screen sizes of mobile devices makes it difficult to create layouts with images that adapt easily from one size to another. In this exercise we'll examine several techniques that can help images work seamlessly from one device to another.

### Objectives

This exercise will focus on

- Image considerations for multiple devices
- Using background images to control resource loading
- Design considerations for multiple screens
- Reducing resource requests with Data URI
- Using CSS Sprites

### Steps

#### Serving alternate images through the use of background images

Using background images can dramatically reduce overhead when moving between desktop and mobile designs. By placing the requests for images in the device's respective styles you limit the images to being downloaded by those devices only. This means you could use larger images for desktop styles while requesting smaller images, or no images at all, for mobile devices. This strategy does require you to consider how images will be used across multiple devices, when it is appropriate to use multiple images, and whether or not the size of the images is a factor in using more than one image.

1. Watch the instructor as he opens the `desolve_mockup.ai` and `banner` Photoshop files and describes the strategy behind how the home and gallery banners will work across multiple layouts. You can find these files in the `Exercise_05/_assets` folder. Note that the **large background images** are **900px x 250px** while the **smaller** versions are **485px x 175px**.
2. Open the `index.htm` from the `Exercise_05` folder.
3. Switch to **Code View** and find and explore the banner section (line 35) and the individual gallery preview sections (beginning on line 41). Note the classes used, and the structure of the elements. Within the banner section, the entire banner div tag will be used to display the large skyline image. For the

individual sections, the class of the parent section will be used to help identify which banner graphic belongs in the preview div tag.

4. In the **desktop.css** file, find the **.banner** selector ( around line 83)
5. Add the following properties to the existing rule:
  - a. height: 350px;
  - b. background: url(../\_images/gallery\_banner.jpg) no-repeat;
6. In the same file, locate the **article .preview** rule (around line 136). Add the following property:
  - a. height: 250px;
7. In the preview rules that follow, add the following properties for each of the following selectors:
  - a. **.philly .preview**
    - i. background: url(../\_images/philly\_banner.jpg) no-repeat;
  - b. **.chicago .preview**
    - i. background: url(../\_images/chicago\_banner.jpg) no-repeat;
  - c. **.nyc .preview**
    - i. background: url(../\_images/nyc\_banner.jpg) no-repeat;
8. Save the file and preview either in the browser or using Live View. Resize the window and note how portions of the images are cut off when resized. While this method doesn't actually scale the images, it does allow us to create flexible "panes" to display images that aren't harmed by the cropping caused by resizing the viewport.
9. Switch focus to the **tablet.css** file.
10. Find the **.banner** selector (around line 57) and add the following property:
  - a. background: url(../\_images/sm\_skyline.jpg) no-repeat;
11. Find the **article .preview** selector (around line 114) and add the following property:
  - a. height: 175px;

12. Below that rule, add the smaller banner images for each of the preview rules:
  - a. **.philly .preview**
    - i. background: url(../\_images/sm\_philly\_banner.jpg) no-repeat;
  - b. **.chicago .preview**
    - i. background: url(../\_images/sm\_chicago\_banner.jpg) no-repeat;
  - c. **.nyc .preview**
    - i. background: url(../\_images/sm\_nyc\_banner.jpg) no-repeat;
13. Save the file and test. Resize the viewport to trigger the tablet styles and notice how the smaller images replace the desktop banners. Resize further to see how the mobile styles use the smaller images. Note that the banner graphic is not used for mobile styles. This reflects the design decision to limit the amount of images requested for the mobile version of the site.

### Using Sprites

Sprites allow you to limit the number of requests you make for images by combining multiple images, such as icons, into a single file. By using background positioning, you can move through the Sprite to display only the icon you need. While larger in file size than an individual icon, savings are realized by making a single request.

14. Watch as the instructor opens up the **icon\_sprites\_25.ai** file and describes the layout and usage of the sprites file. You can find this file in the **Exercise\_04/\_assets** folder if you want to explore the graphic on your own.
15. Based on your personal preference, focus on the **desktop.css** styles in either the related files or CSS Styles panel.
16. Find the **menu-specific** link styles (around line 70 in the code, or the styles beginning with the a.gallery selector)

17. Add the following properties to the individual rules as listed here:

**a. a.gallery**

i. background: url(../\_images/icon\_sprites\_25.png) no-repeat 0 3px;

**b. a.gear**

i. background: url(../\_images/icon\_sprites\_25.png) no-repeat 38px -98px;

**c. a.interact**

i. background: url(../\_images/icon\_sprites\_25.png) no-repeat 62px -197px;

**d. a.shop**

i. background: url(../\_images/icon\_sprites\_25.png) no-repeat 6px -297px;

18. **Save** the file and **preview** it. Note how each of the icons are being displayed based on the background position supplied through the individual rules. Also note that in order for the icons to display properly, the height and left padding are being set through the **#mainNav a** selector.

### Using Data URI to embed images into CSS

Data URI is an inclusion method that allows you to embed inline data in web pages and style sheets. What that means in the real world is that Data URI provides you with a way of embedding images directly into your pages as encoded data, to be processed and displayed by the browser at run-time. While there are pros and cons to the technique, it's a great way to provide smaller images to mobile devices, as it doesn't increase the size of your code by much, and it limits the number of requests made to the server.

19. Open the **contact.htm** and **resources.htm** pages in the **Exercise\_05** folder.

20. Preview the **contact.htm** using **Live View**. The icons for the contact options in the top section are delivered through Data URIs.

21. In the **main.css** style sheet, locate the styles that control the contact-specific links (around 241). Note how the Data URI format uses base64 encoding to store image data. We'll need to generate another icon for the **a.twitter** link.

22. Switch to the **resources.htm** page and preview it page. Click on the online data converter link. This will take you to Websemantics' Image to Data URI Converter, one of the best online data conversion tools I've found.

23. Click the **browse** icon to choose an image. From the **Exercise\_05/\_images** directory, choose the **twitter\_icon.png**. Click **convert** to generate the Data URI code.
24. Scroll down to find the **raw data** code. Highlight and **copy** all this code. Be very careful to copy just the code inside the raw data window, and to make sure you don't miss any characters.
25. Return to the **main.css** file and find the **a.twitter** rule on line 241. Inside the URL parenthesis, first type in **data:image/png;** and then **paste** the copied Data URI code. (If internet connections prevent the generation of the Data URI code, you can find it in the **Exercise\_05/\_assets** directory as **twitter\_icon\_URI.txt**. Simply copy and paste the code from that file in the same manner).
26. After the parenthesis, add **no-repeat left center** to the rule to control the tiling and positioning of the icon.
27. Save the file and preview using **Live View**.

### Using CSS3 to draw buttons and icons

Often the best choice for saving image file size and reducing http requests is to not use an image at all. Several new capabilities within CSS make it possible to replace many of the icons, buttons, and patterns that you have previously served with images with pure CSS.

28. Return to the **index.htm** file found in the **Exercise\_05** folder.
29. Preview the page in Live View. Scroll down in the sidebar until you see the entry for this month's contest. Note that there is a link to enter the contest below the body copy. We'll use CSS to make this link look more like a button.
30. In the **main.css**, find the **.enter a** selector (around line 312)

31. Add the following properties:
  - a. border: 1px;
  - b. margin: 1em 0 1.5em;
  - c. color: rgb(76,67,65);
  - d. width: 5em;
  - e. height: 1.4em;
  - f. line-height: 1.4em;
  - g. text-align: center;
  - h. -webkit-border-radius: .5em;
  - i. -moz-border-radius: .5em;
  - j. border-radius: .5em;
  - k. background: rgb(226, 226, 226)
32. Save and preview in Live View. While it's certainly more "button-like", it's no great replacement for an image just yet. We'll use CSS gradients to further enhance the button.
33. Switch to the **resources.htm** file and preview it in **Live View**.
34. Hold the **CMD** key down and **click** on the **Ultimate CSS Gradient Generator** link.
35. If your internet connection is working, you will browse to Alex Sirota's Ultimate CSS Gradient Generator page. If it's not skip to steps for pasting the gradient code, and open the gradients.txt file from the Exercise\_05/\_assets folder to retrieve the gradient code.
36. After experimenting with the gradient generator and viewing the resulting code, scroll through the presets and **choose the White Gloss #1 preset, which is the second preset from the right, on the second to last row**. Follow the instructor as he does this. If you want to use another gradient, feel free to use another preset or create your own.
37. In the **color format** options below the code, choose **RGB** for color format.
38. **Copy** all of the generated code in the CSS pane. **EXCEPT** for the first line which provides a generic background for older browsers.
39. Tab back to the **index.htm** file, switch to the **main.css** file and **paste** the generated code directly **after** the existing background property for the **.enter a** rule. Save the file and preview in Live View.

40. Next, tab back to the **resources.htm** page. In the **adjustments** area to the left of the generated code, click **reverse** to reverse the direction of the gradient. Check to make sure color is being represented as **RGB**.
41. Once again, copy all of the generated code save for the first line and switch back to the **index.htm** file.
42. **Paste** the reversed code directly **AFTER** the existing background property in the **.enter a:hover** rule.
43. From the **coding toolbar**, choose **Apply Source Formatting** to clean up the pasted code.
44. **Save** the file and **preview** in **Live View**, hover over the generated button to see the effect of reversing the gradient.

While many of the new CSS features such as transforms, gradients, drop shadows, and border-radius allow you to create complex visual effects, you should be careful about how you choose to use them in a mobile context. Many of these properties create additional processing work for the browser, although the savings of not using images for the same task often offsets these concerns.

## Exercise 6: Designing for the Mobile Context

Designing for mobile devices isn't quite the same as designing for desktop devices. Often the context of how the user is using the device is very different. Mobile devices are just that –mobile- and users will often be interacting with content while doing other activities. Connection speeds are often limited by the reach and speed of cellular networks or slower public wifi hotspots. In addition to these physical differences, there are many differences in how people expect to interact with content within the mobile context. Typically users are more task-focused, needing to perform a specific task or retrieve specific information rather than leisurely browsing content. Mobile devices also present you with capabilities that desktop browsers either don't possess or aren't relevant in the desktop context. Geolocation, text messaging, phone and camera integration, and other features allow users to interact with your content in totally new and different ways than previously available to us.

All of this must be taken into consideration as you plan and construct your site. Even if you don't plan to build a native application or separate mobile site, there are things that you can do to take advantage of mobile capabilities and design in a way that is considerate of the mobile user.

### Objectives

This exercise will focus on

- Using styles to present alternate content to mobile devices
- Modifying layouts for mobile context
- Binding touch events for mobile devices
- Taking advantage of mobile-specific features

### Steps

#### Creating alternate content for mobile devices

1. Open the **philadelphia.htm** from the **Exercise\_06** folder.
2. **Preview** the page in **Live View** and using the Window Size picker in the status bar, choose the **Smart Phone** (320 x 480) window size. Scroll down the page and find the menu for the Archived Galleries. While this type of list works well for desktop, it's an inefficient use of space for mobile devices. We'll replace this list with a drop-down menu for mobile devices.
3. Switch to **Code View** and in the Source Code, scroll to **line 87** to find the closing tag of the existing list menu.
4. In the panel dock, click on the **Assets** panel tab to activate that panel. Click on the **library** icon to show library assets.

5. Holding the **Opt** key down, drag the **select** library item to directly **after** the closing **ul** tag for the list menu. Be very careful not to place it within the tag. Holding the option key down will prevent Dreamweaver from adding a link to the library item, and just place the code on the page.
6. Switch to the **mobile.css** stylesheet. We'll need to hide the existing list menu for mobile devices and set the styling for the new select element.
7. Find the **aside nav ul** selector (around line 193). Add the **display: none** property to the rule.
8. Directly underneath that rule, find the **nav select** rule. Add the following properties:
  - a. width: 200px;
  - b. margin-left: 20%
9. Just as we hid the unordered list for mobile devices, we'll need to hide the select element from desktop displays. Switch to the desktop.css file.
10. Scroll down and find the **nav select** selector (around line 266). Add the **display: none** property to the rule.
11. Change back to **Design View** and return the display to full screen. Scroll through the sidebar and note the select menu does not appear. The same rule already is applied to the tablet styles, meaning our select menu will only show up on screen sizes smaller than 481 pixels.

In many cases, JavaScript is used to parse a list of links and generate a select element at runtime. In many cases this approach is appropriate, but often it is also acceptable to utilize CSS to determine which element to serve. Just understand that you are adding additional code that assistive devices and other user agents will still parse. In any event, such select menus will still need to be wired with JavaScript to enable navigation.

## Modifying layouts for mobile context

12. Still in the **philadelphia.htm** file, preview the page using the **Multiscreen Preview** panel. Note the difference in user experience from the desktop and tablet layouts to the phone layout. In particular, note how much scrolling you need to do before getting to the actual gallery images. For the mobile context, we want the intro text to be optional for users, and need to give them a mechanism to be able to read, or ignore the text as they see fit. To do this, we'll use a mixture of CSS and JavaScript. The CSS will allow us to hide or expand the text, while the JavaScript will allow us to respond to touch events common to most Smart Phones.
13. We'll start by giving users an element to trigger the toggling of the text. Switch to **Code View**, target the **Source Code**, and browse to the **intro** text (around line 55).
14. In the **div** element with the class of "**intro**" **add** the class of "**nonTouch**." This class will be used to indicate whether the device supports touch events or not.
15. **After** the first paragraph of text, **create an empty paragraph** with a **class** attribute of "**more**." While non-semantic, this empty paragraph will not render on other layouts, while giving us an element within the DOM to target.
16. Switch to the **mobile.css** file. Find the **.intro** selector (around line 131).
17. In the **.intro** selector, above the commented code, add the following properties:
  - a. height: 155px;
  - b. overflow: hidden;
  - c. margin-bottom: 2em;
18. **Save** the file and switch over to Live View and preview the page using the Smart Phone window size. Note that the intro text is now partially hidden.
19. Below the **.intro** selector, find the **.nonTouch:hover**, **.hover\_effect** selector. **Add** the following properties to the rule:
  - a. height: auto;
  - b. overflow: visible;
20. **Save** the file and preview in Live View again. Hover over the text and note it's expansion.

21. Although the text is expanding in mobile view as desired, there is nothing indicating to the user that they should expect this behavior. We'll use CSS to generate a button icon for us that helps instruct the user.
22. Return to the **mobile.css** styles code. Below the **.nonTouch:hover** selector, you'll find the **.intro .more:after** selector. The **:after** pseudo-element allows us to generate and style content after an element's existing content.
23. From the **Exercise\_05/\_assets** folder, open the **intro\_styles.txt** file.
24. Copy the styles and paste them into the **.intro .more:after** rule. Go through the properties with the instructor to understand how the content is being created and styled.
25. **Save** and **preview** the page in Live View. Note how the button remains even as the text is being hovered over.
26. Return to the **mobile.css** styles and locate the **.nonTouch:hover .more:after** grouped selector. Add the following properties to the rule:
  - a. `content: normal;`
  - b. `display: none;`
27. **Save** and **preview** the page again. Hover over the text and note how the button is removed while the text is being hovered over.

### Binding touch events

For most Smart Phones, there is no such thing as a hover event. While most phone browsers will treat `:hover` styles as a mouseover event, relying on `:hover` styles can cause unexpected behaviors, especially when `:hover` events are combined with click events, such as drop-down menus. Through the use of JavaScript, you can replace `:hover` events with touch events.

28. Switch back to the **philadelphia.htm** file and in **Code View**, click on the **Source Code** to display the page's html.
29. **Below** the existing script tag that links to jQuery (line 17), **add** an opening script tag, hit enter twice and add a closing script tag (**<script></script>**)

30. Inside the new script tags, add the following JavaScript. Follow along with the instructor as he goes through the purpose of the script.

```
$(document).ready(function() {
    //is this a touch device?
    var hasTouch = 'ontouchend' in document ? true : false;
    //if the device is touch enabled, add the hover effect
    class and strip the hover styling for non-touch devices
    if(hasTouch) {
        $('.intro').removeClass('nonTouch');
        $('.more').bind('touchend', function(e) {
            $('.intro').addClass('hover_effect');
        });
    }
});
```

Essentially the script first checks to see if the device is touch enabled. If it is, the nonTouch class is removed from the intro div tag, essentially disabling the :hover styles. Then, a touchend event is bound to the “more” paragraph, making the device listen for the element to be touched. If the element is touched, the class “hover\_effect” is added to the intro div tag, essentially triggering the expansion of the text. Note that in this case the script is a one-time toggle. If we wished to give the user the option to expand or collapse the text, we could add another button to the end of the text when it expands and bind a touch event to it that would collapse the text.

### Using mobile specific features

One of the unfortunate side effects of not designing for mobile as part of the initial design process is missing out on many of the capabilities native to mobile devices that can enhance user experiences. Many of these features can be leveraged with very little effort on the part of the designer, while adding considerable benefit to mobile users. This is especially true of forms, where small changes can lead to usability benefits to your users.

31. Open the **contact.htm** file in the **Exercise\_06** folder.
32. In **Design View**, scroll down to the contact form.
33. **Select** the **Input** element for **Email** and hit the **CMD + T** keyboard shortcut to bring up the Quick Tag editor.
34. From the list that appears for input type, choose “**email**”.
35. Repeat the procedure for the **Personal Website** input, changing it’s type to “**url**”.
36. Scroll down to the **search** form in the footer and change its **type** to “**search.**”

37. **Highlight** the **phone number** and using the **Properties Inspector**, add **tel:8885551234** to the link input to add a link.
38. Use the tag selector to select the new a tag and apply the class "phone" using the Properties Inspector.
39. Preview the page using the Multiscreen Preview and see how the styles change the way that the phone number is presented based on device type.

Watch the instructor as he previews the contact page on a mobile device. Notice the difference in keyboard configuration for the different input types. Note how the telephone link allows the user to make a call to the number just by clicking the link. These are small changes to the contents of the page that have a big impact on user experience on mobile devices.

## Exercise 7: Managing Resources

Designing for mobile is more than just designing for a small screen. You also have to take into consideration the way people connect to those devices, and the limitations of the hardware itself. These limitations mean that resource-intensive sites often perform poorly on mobile devices. Managing your resources properly, and serving mobile devices only the resources they need, can go a long way towards improving the performance of your site across platforms. A good mobile strategy will seek to reduce the amount of http requests a site makes, lower the size and number of images on the page, and eliminate sending resources to the device that it doesn't need.

### Objectives

This exercise will focus on

- Conditionally loading resources based on screen size
- Adding media query support for non-supporting browsers
- Combining resources to limit http requests
- Using Dreamweaver to reduce the overall weight of external resources

### Steps

#### Conditionally loading resources based on screen size

1. Open the **philadelphia.htm** from the **Exercise\_07** folder.
2. Preview the page in a browser, resize the page and notice what happens to the main headline as it switches to the mobile styles. The lettering effect that is being driven by the letter.js library is not used in the mobile space, therefore, there is no need to serve the library to mobile devices.
3. Switch to Code View and locate the script tag that loads jQuery (around line 17).
4. Above that line, add another script tag that loads the Modernizr library:

```
<script src="_scripts/modernizr-custom.js"></script>
```

Modernizr is a JavaScript library that allows you to test for feature support, mainly CSS3 and HTML5 features. By using it's detection features, and taking advantage of the built-in YepNope library support, we'll be able to conditionally load up resources based on the media query used.

5. **Create** a few lines of empty code beneath your new script tag. Eventually all other scripts will be replaced by what we are doing, but for now we'll leave the other scripts in place to pull items from.

6. **Enter** the following code to create a skeleton of the Modernizr load function. If, at any time you find yourself with frustrating coding errors, you can copy and paste these scripts from the **Exercise\_07/assets/conditional\_loading.txt** file:

```
<script>
//use Modernizr.load to determine which external resources to
load and which scripts to run
Modernizr.load([ {

  }
])
</script>
```

7. The **Modernizr.load** function can be used to load up external resources based on tests passed into it.
8. **Inside** the load function, **add** this line of code to load up the jQuery library:

```
load: '_scripts/jquery-1.6.3.min.js',
```

9. In the source code, find the script tag that currently loads the jQuery library and **delete** it.
10. Since Modernizr.load uses asynchronous loading, we need to use a callback function in order to use jQuery. Underneath the load statement, **add** the following code:

```
callback: function (url, result, key) {

} // end callback function
```

11. **Inside** the callback function, we'll move the existing script that handles binding touch events to our text expander. Scroll down in the code and find the script tags that hold the **hasTouch** function. **Cut** the entire function (but not script tags) and **paste** it into the callback function so that it now looks like this:

```
callback: function (url, result, key) {
  //bind touch events to more button
  $(document).ready(function() {
    //is this a touch device?
    var hasTouch = 'ontouchend' in document ? true : false;
    //if the device is touch enabled, add the hover effect class
    and strip the hover styling for non-touch devices
    if(hasTouch) {
      $('.intro').removeClass('nonTouch');
      $('.more').bind('touchend', function(e) {
        $('.intro').addClass('hover_effect');
      });
    };
  }); //end hasTouch function
} // end callback function
```

12. **Delete** the empty script tags where the hasTouch function used to be.
13. Find the closing bracket '}' for the load function. **Add a comma (,)** after it and the following **comment**: //end initial load

```
}, //end initial load
```

14. We now need to run a test to see which media query is being used. If it is not the mobile media query, we'll load the lettering.js library.
15. **Add** the following code just **below** the end initial load bracket:

```
// test to see if screen width is above 481, and then load resources
accordingly
{
  test : Modernizr.mq('only all and (min-width: 481px)'),
  // load lettering.js if it is a larger screen
  yep : { 'bigscreen' : '_scripts/jquery.lettering-0.6.min.js'},
}
}
```

16. The test is where we check for certain features, or in this case, pass a media query for Modernizr to test. In this case, it's checking to see if the screen is larger than 480px. The 'yep' is where we tell Modernizr what to load if that is

the case, note that we're also passing what's known as a 'key' value, the bigscreen value, which we can use to apply conditional logic in our next step. Scroll down through your code and delete the script tag currently importing the lettering.js library.

17. Next we'll need to target the element on the page that we want to use lettering.js with. Again, we'll need to use a callback function to do this. After the yep line, add the following code:

```
// after loading lettering, target the galleryTitle with the
lettering library
callback: function (url, result, key) {
  // if lettering was the file loaded, execute this script
  if (key == 'bigscreen') {
    $(document).ready(function() {
      //apply the lettering function
      $(".galleryTitle").lettering();
    });
  }
}
```

18. The final code should look like this, be very careful to properly close brackets ('{}):

```

<script>
//use Modernizr.load to determine which external resources to load and which
scripts to run
Modernizr.load([ {
load: '_scripts/jquery-1.6.3.min.js',
callback: function (url, result, key) {
//bind touch events to more button
$(document).ready(function() {
//is this a touch device?
var hasTouch = 'ontouchend' in document ? true : false;
//if the device is touch enabled, add the hover effect class and strip the
hover styling for non-touch devices
if(hasTouch) {
$('.intro').removeClass('nonTouch');
$('.more').bind('touchend', function(e) {
$('.intro').addClass('hover_effect');
});
};
}); //end hasTouch function
} // end callback function
}, //end intial load
// test to see if screen width is above 481, and then load resources accordingly
{
test : Modernizr.mq('only all and (min-width: 481px)'),
// load lettering.js if it is a larger screen
yep : { 'bigscreen' : '_scripts/jquery.lettering-0.6.min.js' },
// after loading lettering, target the galleryTitle with the lettering library
callback: function (url, result, key) {
// if lettering was the file loaded, execute this script
if (key == 'bigscreen') {
$(document).ready(function() {
//apply the lettering function
$(".galleryTitle").lettering();
});
}
}
}
])
</script>

```

19. Scroll down and **remove** the script that used lettering js to target the galleryTitle page element. **Save** and test your file. It should still look and function the same as before, however it now only serves the lettering resource to the proper devices based on screen size.

### Adding media query support for non-supporting browsers

Basing so much of our layout and styling on media queries means that we are relying on them to work in order for our pages to be styled properly. Unfortunately, prior to Internet Explorer 9, IE did not support media queries. This leaves a very large hole of potential clients who might not otherwise be able to see our site as we intend. To remedy this, we'll use the respond.js library, a small (3k) library that adds support for media queries.

20. Still in **philadelphia.htm**, return to code view and create a new **script** tag above the script that loads Modernizr. In it, **link** to the **respond.min.js** script.

```
<script src="_scripts/respond.min.js"></script>
```

21. Since we want to only call the respond resource if the browser is IE below version 9, we'll use a conditional comment to screen it from other browsers.
22. Open the **Snippets** panel and **highlight** the **respond.js** script tag.
23. In the **Snippets** panel, open the **Comments** folder, and **double click** the **If Less Than IE 8** conditional comment.
24. In the **comment**, **replace** the **8** with a **9** to target IE versions less than 9.
25. Save the file.

### Combining resources

For mobile devices, additional http requests can dramatically affect performance. So much so that downloading a single large file is often preferable to downloading multiple small files. With that in mind, we'll use inline media query syntax to combine all our CSS files into one file, so that only a single http request is made, regardless of the media query being used.

26. Open the **inline.css** file from the **Exercise\_07/\_css** folder.
27. **Copy** the inline rules from the file. Note that they match the media queries being used in our site.
28. Switch back to **philadelphia.htm** and click on the **main.css** file in the related files list. Make sure you're in **Code View**.
29. Scroll down to the bottom of all styles. **Paste** the inline @media rules at the **bottom** of the **main.css** code.
30. One at a time, **carefully copy** all of the code from the **mobile.css, tablet.css, and desktop.css** and **paste** it into the corresponding **@media section**. Be very careful to stay within the proper @media brackets. **Save** the **main.css** file.
31. Switch back to the **Source Code** for **philadelphia.htm**, and **delete** the **link** tags that link to the **mobile, tablet, and desktop** stylesheets leaving only the link to the main.css. Save all files and test. Your styles should behave exactly as before.

32. Open the **index.htm** and **contact.htm**. **Delete** the **links** to all stylesheets **other** than **main.css**.
33. The **main.css** is now much larger than it was before (33k .vs 18k) but will now only use a single http request.

#### Minimizing code with Dreamweaver

34. Switch to the main.css file.
35. From the menu, go to **Dreamweaver > Preferences** (Mac) or **Edit > Preferences** (PC).
36. Click on the **Code Format** preference.
37. Click the **CSS** button beside Advanced Formatting.
38. **Uncheck** the **Each Property on a Separate Line** checkbox and the **Blank lines between rules** checkbox. Make sure the **All selectors for a rule on the same line** checkbox is **checked**.
39. Click **OK** twice to close the dialogs.
40. In the code toolbar, find the Apply Source Formatting icon (last icon in the toolbar) and choose **Apply Source Formatting**. Notice that you go from 1200 lines of code, to around 300. You could publish this file while reverting the source formatting any time you need to make an edit.
41. **Save** the file.

#### Demo: Page Testing and Conditional image loading

Follow along with the instructor as he demonstrates the conditional resource loading by exploring options for conditional image loading and using the Blaze online testing service to verify resource loading. You can find the fully finished file in the Exercise\_07 finished files folder.

## Exercise 9: Enhancing Mobile Experiences

We'll finish our lab by exploring a few of the "extra" things that we can do to add another layer to our site's mobile experience. Some are crucially important, like the use of the meta viewport tag, while others are merely optional. Taken as a whole, however, these additions to your site can make a big difference for mobile users.

### Objectives

This exercise will focus on

- Setting up a meta viewport tag
- Adding a home page icon
- Preventing text scaling on orientation change

### Steps

#### Setting up a meta viewport tag

Meta viewport tags are extremely important to how your sites are viewed on mobile devices. By using them, you can force the device to report its actual width (rather than its viewport width), control initial scaling, prevent user scaling, and more. We'll use a meta viewport tag to ensure our page shows up at exactly the size and initial scale that we've designed for it.

1. Open the **index.htm** from the **Exercise\_08** folder.
2. In the **Insert Toolbar**, find the **Common** tab and choose the **Head** pulldown tab. Select **Meta** to add a meta tag to the code.
3. In the dialog box, type in **viewport** for **name** and **width=device-width, initial-scale=1.0** for **content**.
4. Go into code view and **move** the **meta** tag **above** the **css** link. **Save** the file.

#### Adding a home page icon

5. Still in Code View, add the following link after the meta tag:

```
<link rel="apple-touch-icon" href="_images/apple-touch-icon.png">
```

6. Save the file. When bookmarked, users will now have the option of adding this launch icon to their interface.

### Preventing text scaling

Although the meta tag forces the initial view to 100%, changing page orientation can often result in unwanted text scaling by mobile devices. To help prevent this on supporting phones, we'll prevent this by adding a single property to our CSS.

7. Click on the **main.css** in the related files list to switch to our styles.
8. Scroll down to our mobile styles (around line 414) and find the grouped selector with the "disable text sizing" comment above it.
9. Add the following property:

```
-webkit-text-size-adjust:none;
```

10. Save the file.

### Demoing Enhancements

Watch the instructor as he demos the new changes to our mobile site.