RESPONSIVE IMAGE MARKUP

Excerpt from:
Learning Web Design, 5e

by Jennifer Robbins

Author's Note: *This article originally appeared in* Chapter 7, Adding Images *in* Learning Web Design, 5e*. It was removed from the 6th edition due to space limitations, but I have made it available here to provide additional detail and practice. If you have the 6th edition, you will find that some text is repeated.*

Pretty quickly after smartphones, tablets, and other devices hit the scene, it became clear that large images that look great on a large screen were overkill on smaller screens. All that image data...downloaded and wasted. Forcing huge images onto small devices slows down page display and may cost real money too, depending on the user's data plan (and your server costs). Conversely, small images that download quickly may be blurry on large, high-resolution screens. Our trusty `img` element with its single `src` attribute just doesn't cut it in many cases.

The solution is to use responsive images, a technique in which you serve multiple versions of the same image, provide information about the images in the markup, then allow the browser to choose and download only the most appropriate file based on what it knows about the user's viewing environment. Screen dimensions are one factor, but resolution, network speed, what's already in its cache, user preferences, and other considerations may also be involved in the brower's selection.

The primary goal is better performance. The investment in writing extra markup ensures that data isn't downloaded unnecessarily. It also provides a better user experience because you can make sure images are appropriate for the screen size and are as crisp as possible.

IN THIS ARTICLE

Width-based image selection using w-descriptors

The srcset and sizes attributes

Screen density-based selection using x-descriptors

The art direction scenario (picture element)

Alternate image file formats (picture element)

You provide multiple images, sized or cropped for different screen sizes, and the browser picks the most appropriate one based on what it knows about the current viewing environment.

RESPONSIVE IMAGE SCENARIOS

The responsive image techniques address four basic scenarios:

- Providing a set of images of various dimensions for use at **different sizes in a responsive layout**

- Providing extra-large images that look crisp on **high-resolution screens**
- Providing versions of the image with varying amount of detail based on the viewport size (known as the **art direction** use case)
- Providing **alternative image formats** that store the same image at much smaller file sizes

In this article, we'll look at the markup for accommodating each of these common use cases.

WIDTH-BASED IMAGE SELECTION (W-DESCRIPTOR)

When you're designing a responsive web page, chances are you'll want images to resize based on layout or viewport. If you see an image with a significant difference between its smallest and largest dimensions in the layout, that should be a clue that the image would benefit from the width-based responsive image technique.

For a browser to download only the most appropriately-sized image from a set of provided versions, it needs to know:

- What images are available and their actual (intrinsic) widths
- What size the image (`img`) will appear in the layout

We can provide that information right in the `img` element with the `srcset` and `sizes` attributes, respectively.

The srcset Attribute

The `srcset` attribute provides a list of available image files for the browser to choose from. Its value is a comma-separated list of options. Each item in the list has two parts: the location (URL) of the image, and a width descriptor (indicated by a `w`, also commonly referred to as a w-descriptor) that provides the actual pixel width of the image. Using `srcset` with a w-descriptor is appropriate when the images are identical except for their dimensions (in other words, they differ only in scale).

Note that the whole list is the value of `srcset` and goes inside a single set of quotation marks. This sample shows the structure of a `srcset` attribute and its values:

```
srcset="image1-URL #w, image2-URL #w"
```

Here's an example of a `srcset` attribute that provides four image options and specifies their respective pixel widths via w-descriptors. I've stacked the options here to make them easier to read. Note again that the whole list is in a single set of quotation marks:

```
srcset="strawberries-480.jpg 480w,
        strawberries-960.jpg 960w,
        strawberries-1280.jpg 1280w,
        strawberries-2400.jpg 2400w"
```

The sizes Attribute

The sizes attribute is required when you use w-descriptors.

That's a good start, but whenever you use w-descriptors, you also need to include the **sizes** attribute that tells the browser what size the image element will occupy in the layout at various viewport widths. There is a very good reason (in addition to being required in the spec), and it's worth understanding.

When a browser downloads the HTML document for a web page, the first thing it does is look through the whole document and establish its outline structure (its Document Object Model, or DOM). Then, almost immediately, a preloader goes out to get all the images from the server so they are ready to go. Finally, the CSS and the JavaScript are downloaded. It's likely that the style sheet has instructions for layout and image sizes, but by the time the browser sees the styles, the images are already downloaded. For that reason, we have to give the browser a hint with the **sizes** attribute whether the image will fill the whole viewport width or only a portion of it. That allows the preloader to pick the correct image file from the **srcset** list.

We'll start with the simplest scenario in which the image is a banner and always appears at 100% of the viewport width, regardless of the device (FIGURE A). Here's the complete **img** element with the **srcset** and **sizes** attributes (note that the **src** attribute is still required and is used to provide a fallback image):

```
<img src="strawberries-640.jpg"
     alt="baskets of ripe strawberries"
     srcset="strawberries-480.jpg 480w,
             strawberries-960.jpg 960w,
             strawberries-1280.jpg 1280w,
             strawberries-2400.jpg 2400w"
     sizes="100vw">
```

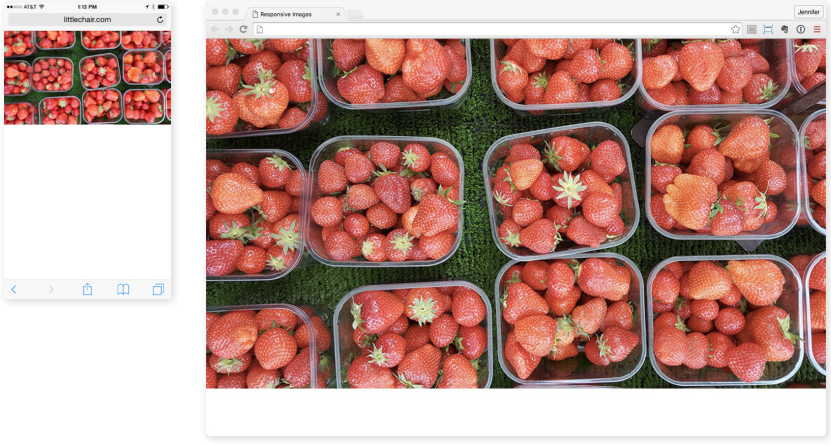


FIGURE A. The image fills 100% of the viewport width, regardless of its size.

Browsers that do not support srcset and sizes use the image specified in the src attribute.

In this example, the **sizes** attribute tells the browser that the image fills the full viewport by using viewport width units (**vw**), the most common unit for the **sizes** attribute, so the browser can pick the best image for the job. For example, **100vw** translates to 100% of the viewport width, **50vw** would be 50%, and so on. You can also use **em**, **px**, and a few other CSS units, but you cannot use percentages. Browsers that do not support **srcset** and **sizes** simply use the image specified in the **src** attribute.

Sizing an image to fill the whole width of the browser is a pretty specific case. More likely, your images will be one component in a responsive page layout that resizes and rearranges to make best use of the available screen width. FIGURE B shows a sidebar of food photos that take up the full width of the screen on small devices, take up a portion of the width on larger devices, and appear three across in a layout for large browser windows.

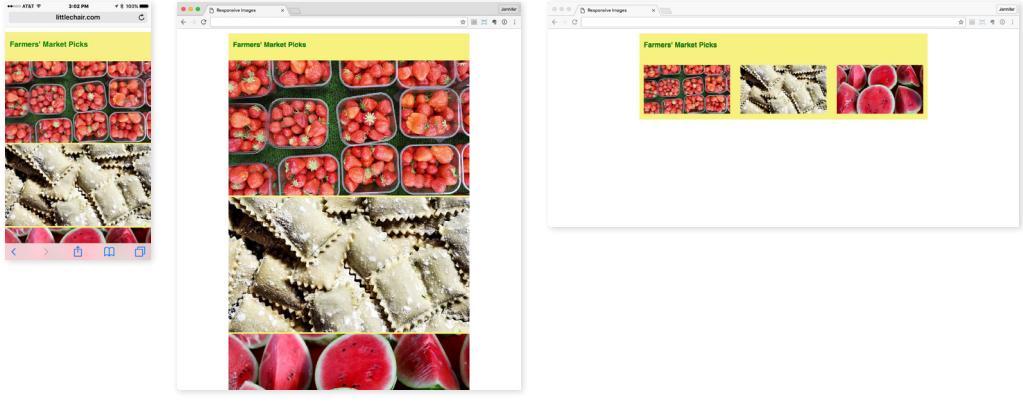


FIGURE B. The width of the images changes based on the width of the viewport.

For cases like these, use the **sizes** attribute to tell the browser something about how the image will be sized for each layout. The **sizes** value is a comma-separated list in which each item has two parts. The first part in parentheses is a media condition (like those used in media queries) that describes a parameter such as the width of the viewport. The second part is a length that indicates the width that image will occupy in the layout if the media condition is met. Here's how that syntax looks:

```
sizes="(media-feature: condition) length,
       (media-feature: condition) length,
       default_width"
```

I've added some media conditions to the previous example, and now we have a complete valid **img** element for the first image in FIGURE B:

```
<img src="strawberries-640.jpg" alt="baskets of ripe strawberries"
    srcset="strawberries-240.jpg 240w,
            strawberries-480.jpg 480w,
            strawberries-672.jpg 672w"
     sizes="(max-width: 480px) 100vw,
            (max-width: 960px) 70vw,
            240px">
```

The **sizes** attribute tells the browser the following:

- If the viewport is 480 pixels wide or smaller (maximum width is 480 pixels), the image fills 100% of the viewport width.
- If the viewport is wider than 480 pixels but no larger than 960 pixels (**max-width: 960px**), then the image will appear at 70% of the viewport. (This layout has 15% margins on the left and the right of the images, or 30% total.)
- If the viewport is larger than 960 pixels and doesn't meet any of the prior media conditions, the image gets sized to exactly 240 pixels.

When the HTML loads, the browser checks the width of the viewport and how big the image will appear within it. It can then select the most appropriate image from the **srcset** list to download.

WARNING

The **sizes** *attribute will resize an image even if there is no CSS applied to it. If there is a CSS rule specifying image size that conflicts with the value of the* **sizes** *attribute, the style rule wins (i.e., it overrides the* **sizes** *value).*

How Many Images Do You Need?

One part of the responsive image technique is to provide the set of images at a range of sizes, but how many and what sizes do you need? Unfortunately, the answer is not straightforward due the endless combinations of image sizes and screen densities, and because browsers vary in how they pick from the available options. However, there are a few strategies to use as guidelines.

Largest, Smallest, and In-between

If the size range for your image isn't that large, you might find that providing small, medium, and large versions is fine. If there is a large difference, images for more breakpoints may be required. Or if there is very little difference between endpoints, one image may suffice. Keep in mind that browsers don't know your logic for the image sizes you created; they just pick what they determine to be the best option. The upshot is you don't need to provide an image sized precisely for each breakpoint in the design. A little scaling up or down is acceptable.

Images Based on File Size

Providing a range of selections based on file size, not pixel dimensions, may be a more appropriate approach. With this strategy, the images in the set step up in fixed file size increments, such as 20 KB or 40 KB. Thankfully, there is tool that will generate image sets based on file size for you. The Responsive Image Breakpoints Generator by Cloudinary (*responsivebreakpoints.com*) lets you upload a large image, set the maximum/minimum dimensions, the size step, and the maximum number of images, and it generates all the images automatically.

The RespImageLint Bookmarklet

If you have a working prototype of your page (including **img** elements that have already been marked up with **srcset** and **sizes**), you can use the RespImageLint bookmarklet created by Martin Auswöger to generate the optimum **sizes** value for you. It works by running a script that resizes your page behind the scenes and calculates the actual width of the image on a range of viewport widths. The new **sizes** value it generates can be used to replace your initial value and acts as a guide for generating the images at in-between sizes that make mathematical sense.

Get the RespImageLint bookmarklet at *ausi.github.io/respimagelint/*. Masa Kudamatsu provides an excellent tutorial on using RespImageLint in a simplified responsive image workflow in his article "Responsive Images: DIY Implementation in 6 Steps" (*medium.com/web-dev-survey-from-kyoto/responsive-images-diy-implementation-in-6-steps-a4342ecbb08*).

There's a bit more to using **sizes** than shown here—other media conditions, additional length units, even the ability to ask the browser to calculate widths for you. If you plan on using viewport-width-based images in your designs, I recommend reading the spec to take full advantage of the possibilities.

SCREEN DENSITY-BASED IMAGE SELECTION (X-DESCRIPTOR)

Using **srcset** with w-descriptors is by far the most common responsive method because image sizes tend to be flexible in responsive layouts. However, if you have an image that stays the same dimensions in the layout, and you want to swap it out based solely on the device pixel ratio (DPR) of screen, use **srcset** with an x-descriptor that indicates the target screen density. The **sizes** attribute is not necessary in this scenario.

Before we get to the code, it will be useful to be familiar with device pixel ratios and how images are displayed on screens.

Devices use a measurement called a reference pixel for layout purposes.

Device-pixel-ratios

Everything that you see on a screen display is made up of little squares of colored light called pixels. We call the pixels that make up the screen itself device pixels (you'll also sometimes see them referred to as hardware pixels or physical pixels). Until recently, screens commonly fit 72 or 96 device pixels in an inch (now 109 to 160 is the norm). The number of pixels per inch (ppi) is the resolution of the screen.

Bitmapped images, like JPEG, PNG, and GIF, are made up of a grid of pixels too. It used to be that the pixels in images mapped one-to-one with the device pixels. An image 100 pixels wide would be laid out across 100 device pixels. Nice and straightforward.

It should come as no surprise that it's not so straightforward today. Manufacturers have been pushing screen resolutions higher and higher in an effort to improve image quality. The result is that device pixels have been getting smaller and smaller, so small that our images and text would be illegibly tiny if they were mapped one-to-one.

To compensate, devices use a measurement called a reference pixel (or logical pixel) for layout purposes. Reference pixels are also known as points (PT) in iOS, Device Independent Pixels (DP or DiP) in Android, or CSS pixels because they are the unit of measurement we use in style sheets. A device's viewport is measured in reference pixels, independent of how many physical device pixels make up the screen.

The ratio of the number of device pixels to reference pixels is called the device pixel ratio (DPR) (FIGURE C). Common device pixel ratios on handheld devices are 1x, 1.5x, 2x, 2.4x, 3x, and even 4x (the "x" is the convention for indicating DPR). Even large desktop displays now feature ratios of 2x, 3x, and 4x (see the sidebar "Prioritize 2x Displays").

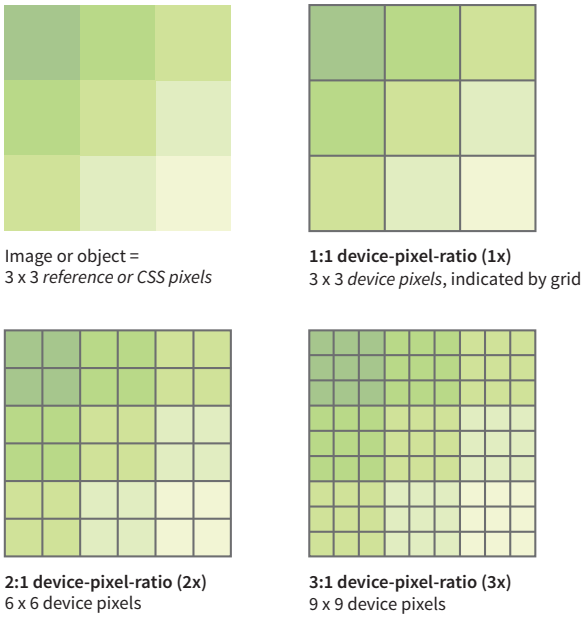


FIGURE C. Device pixels compared to CSS/reference pixels.

Let's say you have an image that you want to appear 200 pixels wide on all displays. You can make the image exactly 200px wide, and it will look fine on standard-resolution displays, but it might be a little blurry on high-resolution displays. To get that image to look sharp on a display with a device-pixel-ratio of 2x, you'd need to make that same image 400 pixels wide and place it in an `img` element with a width of 200px. It would need to be 600 pixels wide to look as sharp as possible on a 3x display. Unfortunately, the larger images may have file sizes that are four or more times the size of the original. Who wants to send all that extra data to a 1x device that really only needs the smaller image?

Providing Options with srcset

The `srcset` attribute can also be used to provide a set of image options based on screen density. When using `srcset` in the `img` element for this purpose, provide the location (URL) of each image along with an x-descriptor that indicates the target device-pixel-ratio for the image (see **Warning**).

Let's look at an example. I have an image of a turkey that I'd like to appear 200 pixels wide. For standard resolution, I created the image at 200 pixels wide

Prioritize 2x Displays

At this point, the vast majority of smartphones have densities of 2x or higher, and the percentage of laptop and desktop monitors with high-density displays is growing. For that reason, it is time well invested to make sure that important photos and graphics look crisp at higher screen resolutions by providing images with higher pixel dimensions.

However, there is evidence that for photographic images, the human eye cannot detect the quality difference between 2x and 3x, so it generally isn't worth the extra bytes to create a version of the image at 3x or higher. In fact, some designers find that images created at 1.5x look good enough on 2x displays. For UI elements, logos, and other small images with crisp edges, you might decide that a 3x or 4x version helps the image look as sharp as the surrounding text, but then again, those types of images are better handled with vector-based SVGs.

The upshot is that when creating image sets for a responsive image, it is recommended that you provide a version for 2x displays but probably not any larger.

WARNING

*You can use w-descriptor **or** x-descriptors with `srcset`, but you cannot use a combination. You'll need to choose one depending on whether accommodating changing image dimensions or screen density is your priority.*

and named it *turkey-200px.jpg*. I'd also like it to look crisp in high-resolution displays, so I have two more versions: *turkey-400px.jpg* (for 2x) and *turkey-600px.jpg* (for 3x). Here is the markup for adding the image and indicating its high-density equivalents with x-descriptors:

```
<img src="/images/turkey-200px.jpg" alt=""
     srcset="/images/turkey-400px.jpg 2x,
             /images/turkey-600px.jpg 3x" >
```

Browsers check the screen resolution and download what they believe to be the most appropriate image. If the browser is on a device with a 2x display, it may download *image-400px.jpg*. If the device-pixel-ratio is 1.5x, 2.4x, or something else, it checks the overall viewing environment and makes the best selection. It is important to know that when we use **srcset** with the **img** element, we are handing the keys to the browser to make the final image selection.

`<picture>…</picture>`
Specifies a number of image options

`<source>…</source>`
Specifies alternate image sources

ART DIRECTION (THE PICTURE ELEMENT)

So far, we've looked at image selection based on the size of the viewport and the resolution of the screen. In both of these scenarios, the content of the image does not change but merely resizes.

But sometimes, resizing isn't enough. You might want to crop into important details of an image when it is displayed on a small screen. You may want to change or remove text from the image if it gets too small to be legible. Or you might want to provide both landscape (wide) and portrait (tall) versions of the same image for different layouts.

For example, in FIGURE 7-D, the whole image of the table as well as the dish reads fine on larger screens, but at smartphone size, it gets difficult to see the delicious detail. It would be nice to provide alternate versions of the image that make sense for the browsing conditions.

This scenario is known as an art-direction-based selection and it is accomplished with the **picture** element. The **picture** element has no attributes; it is just a wrapper for some number of **source** elements and an **img** element. The **img** element is required and must be the last element in the list. If the **img** is left out, no image will display at all because it is the piece that is actually placing the image on the page. Let's look at a sample **picture** element and then pick it apart:

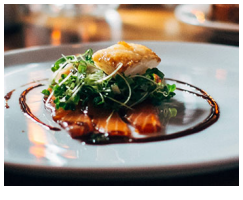
```
<picture>
  <source media="(min-width: 1024px)" srcset="icecream-large.jpg">
  <source media="(min-width: 760px)" srcset="icecream-medium.jpg">
  <img src="icecream-small.jpg" alt="hand holding ice cream cone and
text that reads Savor the Summer">
</picture>
```

This example tells the browser that if the viewport is 1024 pixels wide or larger, use the large version of the ice cream cone image. If it is wider than

760 pixels (but smaller than 1024, such as on a tablet), use the medium version. Finally, for viewports that are smaller than 760 pixels and therefore don't match any of the media queries in the previous **source** elements, the small version should be used (FIGURE 7-E). The small version, as specified

That dinner looks delicious on desktop browsers.
(1280px wide)

Detail is lost when the full image is shrunk down on small devices.
(300px wide)

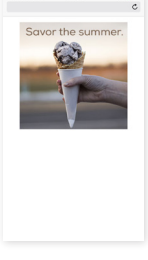


Cropping to the most important detail may make better sense.
(300px wide)

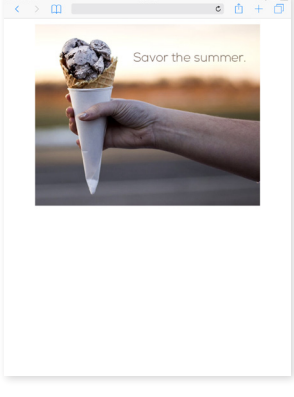
FIGURE D. Some images are illegible when resized smaller for mobile devices.

FIGURE E. The **picture** element provides different image versions to be sourced at various screen sizes.

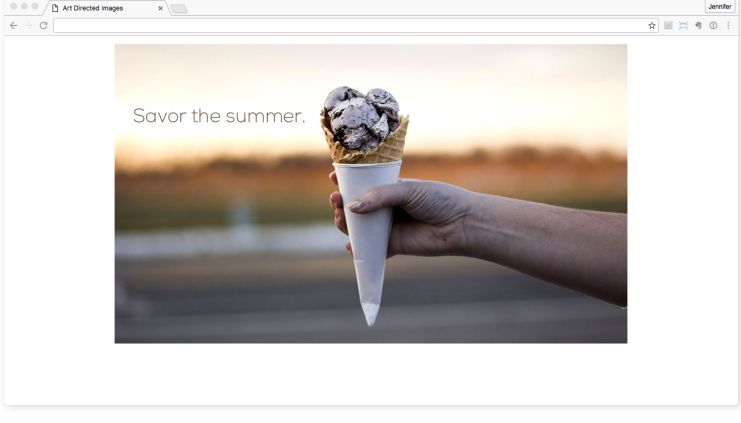
iPhone



iPad



Chrome browser on desktop



in the **img** element, will be used for browsers that do not recognize **picture** and **source**.

Each **source** element includes a **media** attribute and a **srcset** attribute. It can also use the **sizes** attribute, although that is not shown in the previous example. The **media** attribute supplies a media query for checking the current browsing conditions. It is similar to the media conditions we saw in the earlier **srcset** example, but the **media** attribute specifies a full-featured CSS media query. The **srcset** attribute supplies the URL for the image to use if the media query is a match. In the previous example, there is just one image specified, but it could also be a comma-separated list if you wanted to provide several options using w- or x-descriptors.

Browsers download the image from the first **source** that matches the current conditions, so the order of the **source** elements is important. The URL provided in the **srcset** attribute gets passed to the **src** attribute in the **img** element. Again, it's the **img** that places the image on the page, so don't omit it. The **alt** attribute for the **img** element is required, but **alt** is not permitted in the **source** element.

Art direction is the primary use case for the **picture** element, but let's look at one more thing it can do to round out our discussion on responsive images.

ALTERNATIVE IMAGE FORMATS (THE TYPE ATTRIBUTE)

Once upon a time, in the early 1990s, the only image type you could put on a web page was a GIF. JPEGs came along not long after, and we waited nearly a *decade* for reliable browser support for the more feature-rich PNG format. It takes a notoriously long time for new image formats to become universally supported. In the past, that meant simply avoiding newer formats.

In an effort to reduce image file sizes, more efficient image formats have been developed—such as WebP, AVIF, and JPEG-XL—that can compress images significantly smaller than their JPEG and PNG counterparts. And once again, some browsers support them and some don't. We can use the **picture** element to serve the newer image formats to browsers that can handle them, and a standard image format to browsers that can't. We no longer have to wait for universal browser support.

In the following example, the **picture** element specifies two image alternatives before the fallback JPEG listed in the **img** element:

```
<picture>
  <source type="image/avif" srcset="pizza.avif">
  <source type="image/wwbp" srcset="pizza.webp">
  <img src="pizza.jpg" alt="photo of a pepperoni pizza">
</picture>
```

For image-format-based selections, each **source** element has two attributes: the **srcset** attribute that we've seen before, and the **type** attribute for specifying the type of file (also known as its MIME type, see the "File (MIME) Types" sidebar). In this example, the first **source** points to an image that is in the AVIF format, and the second specifies a WebP. Again, the browser uses the image from the first source that matches the browser's image support, so it makes sense to put them in order from smallest to largest file size.

WRAPPING UP RESPONSIVE IMAGES

This has been a long discussion about responsive images, and we've really only scratched the surface. We've looked at how to use the **img** element with **srcset** and **sizes** to make *pixel-ratio-based* and *viewport-size-based* selections. We also saw how the **picture** element can be used for *art-direction-based* and *image-type-based* selections.

I've kept my examples short and sweet, but know that it is possible to combine techniques in different ways, often resulting in a tower of code for each image. Because the code for each responsive image can get a bit cumbersome, many developers automate the responsive image process by using an Image CDN that generates images at the appropriate sizes on the fly.

If you'd like to get a deeper understanding of reponsive image markup and strategies, I recommend the following resources:

- "The Ultimate Guide to Responsive Images on the Web," by Anna Monus (*debugbear.com/blog/responsive-images*)
- The 10-part "Responsive Images 101" tutorial by Jason Grigsby (*cloudfour.com/thinks/responsive-images-101-definitions/*).
- "Responsive Images Done Right: A Guide to picture and srcset," by Eric Portis (*smashingmagazine.com/2014/05/responsive-images-done-right-guide-picture-srcset/*)
- "What Is an Image CDN—the Complete Guide," by Rahul Nanwani (*imagekit.io/blog/what-is-image-cdn-guide/*)

File (MIME) Types

The web uses a standardized system to communicate the type of media files being transferred between the server and browser. It is based on MIME (Multipurpose Internet Mail Extension), which was originally developed for sending attachments via email. Every file format has a standardized type (such as **image**, **application**, **audio**, or **video**), subtype that identifies the specific format, and one or more file extensions. In our example, the **type** attribute specifies the WebP option with its type/subtype (**image/webp**) and uses the proper file extension (*.webp*). Other examples of media MIME types are **image/jpeg** (extensions *.jpg*, *.jpeg*), **video/mpeg** (extensions *.mpg*, *.mpe*, *.mpeg*, *.m1v*, *.mp2*, *.mp3*, and *.mpa*), and **application/pdf** (*.pdf*). The complete listing of registered MIME types is published by the IANA (Internet Assigned Numbers Authority) at *www.iana.org/assignments/media-types*.