SVG stands for [Scalable Vector Graphics](http://en.wikipedia.org/wiki/Scalable_Vector_Graphics" \t "_top), and the name basically says it all. SVG is a [vector-based](http://en.wikipedia.org/wiki/Vector_graphics" \t "_top)graphics system, where you can scale images to any desired size, without any loss in quality. While bitmap graphic formats (like JPEG or PNG) store image data as a series of pixels, vector formats like SVG instead store image data geometrically, as a series of lines, curves, and/or shapes. Because image data isn't tied to a specific number of pixels, the graphic resolution can be as high as the screen allows. Like [MathML](http://chimera.labs.oreilly.com/books/1234000000770/ch04.html), SVG is an XML vocabulary, and can be styled with CSS and scripted with JavaScript, which opens the door to using SVG for animations, interactive graphics, and much more.

While there has been longstanding support for SVG graphics in major Web browsers for a long time, SVG is formally a part of the HTML5 specification, and is now a first-class citizen in HTML5 pages. In this chapter, you'll get a brief introduction to SVG, put together an interactive SVG coloring book, and learn more about SVG compatibility in the ereader ecosystem.

**A Ten-Minute Introduction to SVG**

In order to become familiar with SVG, let's start by building an SVG image from scratch. We'll take our Canvas "smiley" image from [“Drawing on your <canvas>”](http://chimera.labs.oreilly.com/books/1234000000770/ch01.html#canvas_smiley) and reimplement it as an SVG image. Here are the SVG elements we'll use:

<svg>

All SVG images must have a root element of <svg>, which contains all the other elements you use to compose your graphic. The <svg> element should have the namespace declarationxmlns="http://www.w3.org/2000/svg" and a version attribute (in this chapter, we're writing SVG 1.1) to ensure browsers and reading systems parse the content properly:

**<svg** xmlns="http://www.w3.org/2000/svg" version="1.1"**>** *<!-- All your SVG content will go here -->* **</svg>**

<rect>

As its name suggests, [<rect>](http://www.w3.org/TR/SVG/shapes.html" \l "RectElement" \t "_top) is used to construct a rectangle shape. The four key attributes you need to supply on <rect> are:

x

The *x*-coordinate of the top-left corner of the rectangle, assuming the origin is the top-left corner of the page and has the coordinates (0,0)—with positive values moving to the right of the origin, and negative values moving to the left of the origin.

y

The *y*-coordinate of the top-left corner of the rectangle, assuming the origin is the top-left corner of the page and has the coordinates (0,0)—with positive values moving below the origin, and negative values moving above the origin.

width

The rectangle's width (horizontal)

height

The rectangle's height (vertical)

Both x and y are optional, and will default to 0 if omitted. If you omit width or height, the rectangle will not be drawn at all.

Other optional attributes you can supply on <rect> that are not specific to the element arestroke (which lets you specify the color of the rectangle border), stroke-width (the thickness of the rectangle border), and fill (the color that fills the rectangle interior; specify transparentfor a transparent interior).

Here is a sample <rect> element that will draw a rectange with top-left-corner coordinates of (10, 50), width of 60, height of 30, a blue border of thickness 3px, and a yellow interior:

**<rect** x="10" y="50" width="60" height="30" stroke="blue" stroke-width="3px" fill="yellow" **/>**

The shape will render like this (10 pixels right and 50 pixels down from the top-left corner of the page):

http://orm-chimera-prod.s3.amazonaws.com/1234000000770/images/svg_rect.png

<circle>

Again, intuitively, [<circle>](http://www.w3.org/TR/SVG/shapes.html" \l "CircleElement" \t "_top) draws a circle. <circle> takes three attributes:

cx

The *x*-coordinate of the *center* of the circle, assuming the origin is the top-left corner of the page and has the coordinates (0,0)—with positive values moving to the right of the origin, and negative values moving to the left of the origin.

cy

The *y*-coordinate of the *center* of the circle, assuming the origin is the top-left corner of the page and has the coordinates (0,0)—with positive values moving below the origin, and negative values moving above the origin.

cr

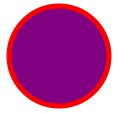
The length of the circle's radius.

Both cx and cy are optional, and will default to 0 if omitted. If you omit cr, the circle will not be drawn at all. As with <rect>, you can specify stroke, stroke-width, and fill attributes on a<circle> element.

Here is a sample <circle> element that will draw a circle with center coordinates (200, 200), radius of 50, a red border of thickness 5px, and a purple interior.

**<circle** cx="200" cy="200" r="50" stroke="red" stroke-width="5px" fill="purple"**/>**

The shape will render like this (with the center 200 pixels from the top and left of the page):



<path>

[<path>](http://www.w3.org/TR/SVG/paths.html" \t "_top) is an extremely flexible SVG element that allows you to draw your own custom shapes by specifying the coordinates and lengths of its component lines and arcs. The key attribute <path>takes is d, which contains all the path data for the shape's outline. There are numerous types of instructions you can supply in a d attribute. Here are just some of the instructions available:

M

A "moveto" instruction, which takes an *x*- and *y*-coordinate where the cursor will be moved to instate a new starting point for the next drawing instruction. The following "moveto" will reposition the cursor at (50,50):

M50,50

H

A horizontal "lineto" instruction, which draws a horizontal line from the current cursor position to a specified *x*-coordinate. For example, if the current postion is (30, 30), the following instruction:

H10

will draw a horizontal line from (30,30) to (10,30). The location (10,30) will become the new cursor position.

V

A vertical "lineto" instruction, which draws a vertical line from the current cursor position to a specified *y*-coordinate. For example, if the current postion is (30, 30), the following instruction:

V10

will draw a vertical line from (30,30) to (30,10). The location (30,10) will become the new cursor position.

L

A more generic "lineto" instruction, for lines that are neither vertical nor horizontal. Ldraws a line from the current cursor position to a specified *x*- and *y*-coordinate. For example, if the current postion is (30, 30), the following instruction:

L10,10

Will draw a line from (30,30) to (10,10). The location (10,10) will become the new cursor position.

A

An "elliptical arc" instruction. Will draw an arc from the current cursor position, based on several parameters separated by spaces, specified in this order: the *x*-radius and the *y*-radius (separated by a comma), the *x*-axis rotation (in degrees), the "large-arc-flag" and the "sweep-flag" (separated by a comma; both parameters take a value of 0 or 1), and the ending-point *x*-coordinate and the ending-point *y*-coordinate (separated by a comma). The "large-arc-flag" and the "sweep-flag" indicate how to draw the arc between the starting and ending point. If the "large-arc-flag" is 0, the smaller-possible arc between the two points will be drawn; if "large-arc-flag" is 1, the larger-possible arc between the two points will be drawn. If "sweep-flag" is 0, the arc will be drawn at a "negative-angle" direction (e.g., sweeping through the bottom-left space between the points); if "sweep-flag" is 1, the arc will be drawn in a "positive-angle" direction (e.g., sweeping through the top-right space between the points).

Detailed discussion of the mathematics and SVG mechanics of elliptical arcs is beyond the scope of this book, but for a good introductory tutorial with a bit more depth, see this [elliptical arc tutorial on Illinois State's site](http://www.itk.ilstu.edu/faculty/javila/SVG/SVG_drawing1/elliptical_curve.htm" \t "_top).

The following instruction—where we start at (0,0) and end at (30,30)—will draw the upper-right quadrant of a circle with a radius of 30.

M0,0 A30,30 0 0,1 30,30

All *x*- and *y*-coordinates referenced above are considered to be *absolute*, which means they are relative to an origin at the top-left corner of the page with the coordinates (0,0)—with positive *x*- and *y*- values moving to the right and down, respectively, and negative*x*- and *y*- values moving to the left and up, respectively.

If you'd prefer to use coordinates that are *relative*, based on the current cursor position and not the origin, you can use the lowercase version of any of the above commands instead—i.e., m, h, v, l, or a.

As with <rect> and <circle>, the <path> element can take stroke, stroke-width, and fillattributes.

Putting together all of the above, here's a <path> element that will draw the top half of a circle centered at (100,100) with a radius of 20, a black border with a thickness of 2px, and a green fill:

**<path** d="M80,100 A20,20 0 0,1 120,100 H80" stroke="black" stroke-width="2px" fill="green"**/>**

This shape will render like this (with the bottom left corner 80 pixels to the right and 100 pixels down from the top-left of the page):

http://orm-chimera-prod.s3.amazonaws.com/1234000000770/images/path_svg.png

<text>

The [<text>](http://www.w3.org/TR/SVG/text.html" \l "TextElement" \t "_top) element is used to add text characters to the image. While <text> can accept a myriad of attributes for fine-tuned placement and rendering of each character in a block of text, in its simplest form, you can just supply the following two attributes to indicate the placement of the text:

x

The *x*-coordinate for the starting point of the text. This coordinate is *absolute*, relative to the origin of the page (0,0).

y

The *y*-coordinate for the starting point of the text. This coordinate is *absolute*, relative to the origin of the page (0,0).

If you choose, you can supply multiple coordinates for either x or y (separated by a comma or space), in which case the *n*th coordinate in the list will indicate the placement of the *n*th character in the text to be rendered.

<text> also accepts [several attributes related to the text font](http://www.w3.org/TR/SVG/text.html" \l "FontPropertiesUsedBySVG" \t "_top), including font-family (font family to use), font-size (font size; supports both absolute and relative sizes), and font-style (e.g., italic)

The text to be rendered should be included inside the <text> tag.

The following example prints the text "I just learned SVG!" at the position (50,50) in an italic serif font:

**<text** x="50" y="50" font-family="serif" font-style="italic"**>** I just learned SVG**</text>**

As you've seen in this section, once you start doing complex drawings in SVG, the markup quickly gets very intricate and potentially quite difficult to write manually. Most people who create SVG graphics do not write the markup by hand; instead, they use a drawing program that can output to SVG format. If you're interesting in creating your own SVG images, you may want to take a look at [Inkscape](http://inkscape.org/" \t "_top), an open source editor that specializes in SVG.

**An SVG "Smiley"**

The SVG basics covered in [“A Ten-Minute Introduction to SVG”](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#svg_intro) are all we need to know in order to redraw our smiley in native SVG. Here's the code, with each line preceded by an annotation describing its function:

**<svg** xmlns="http://www.w3.org/2000/svg" version="1.1"**>** *<!-- To start, draw a border -->* **<rect** x="10" y="10" width="200" height="225" stroke="black" fill="transparent" stroke-width="1px"**/>** *<!-- Now, draw face: a circle at (110,110) with radius of 75, a black border,*   *and a yellow fill -->* **<circle** cx="110" cy="110" r="75" stroke="black" fill="yellow"**/>** *<!-- Now draw left eye: a circle at (75,80) with radius of 10, a black border*   *and a black fill -->* **<circle** cx="75" cy="80" r="10" stroke="black" fill="black"**/>** *<!-- Now draw right eye: a circle at (145,80) with radius of 10, a black border*   *and a black fill -->* **<circle** cx="145" cy="80" r="10" stroke="black" fill="black"**/>** *<!-- Now draw smile: an arc starting at (74,130) with an x/y radius of 35,*   *and ending at 149,130 -->* **<path** d="M74,130 A35,35 0 0,0 149,130" stroke="black" stroke-width="6px" fill="transparent"**/>** *<!-- Smiley speaks: write text starting at (60,210) -->* **<text** x="60" y="210" font-family="sans-serif"**>**Hello SVG!**</text>** **</svg>**

First, we use a <rect> element to draw a rectangular, black border that will contain the smiley. Then we draw a large <circle> (yellow, with black border) representing the smiley's face. Next, we draw two smaller <circle>s (filled in black) for the eyes, followed by a <path> that contains an arc for the actual smile. Lastly, we write "Hello SVG" below the face.

[Figure 5-1](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#hello_svg) shows the masterful result, rendered in Safari for Mac.



*Figure 5-1. Smiley says, "Hello SVG!"*

Now that we've mastered the SVG fundamentals, in the next section, we'll explore how to use JavaScript to dynamically modify SVG content in order to create an interactive graphic.

**An SVG Coloring Book**

In [“Canvas Finger Painting”](http://chimera.labs.oreilly.com/books/1234000000770/ch01.html#finger_painting_section), we took advantage of the interactivity possible with <canvas> to make a simple drawing program for the screen. As a variation on this theme, let's implement a "coloring book" in SVG, where instead of drawing on a black canvas, we'll instead fill in the sections of an existing sketch. Because [everyone loves cats](http://icanhas.cheezburger.com/" \t "_top), we'll take the public-domain SVG shown in [Figure 5-2](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#baseline_cartoon_cat)(available from the [Open Clip Art Library](http://openclipart.org/" \t "_top)), and adapt it for our coloring book.



*Figure 5-2. Sleepy cat SVG (public domain, by Gerald G). Available from Open Clip Art Library: [http://openclipart.org/people/Gerald\_G/Gerald\_G\_Cartoon\_Cat\_Sleeping.svg](http://openclipart.org/people/Gerald_G/Gerald_G_Cartoon_Cat_Sleeping.svg" \t "_top)*

[Example 5-1](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#orig_svg_markup) shows brief snippet of the SVG XML markup that underlies the image in [“Canvas Finger Painting”](http://chimera.labs.oreilly.com/books/1234000000770/ch01.html#finger_painting_section), which consists largely of some quite complex <path>s, so you can get an idea of what the code looks like (the full contents is more than 500 lengthy, jam-packed lines).

*Example 5-1. Snippet of sleepy cat SVG XML*

**<g** id="g5"**>** **<path** d="M 100.885,160.976 C 100.375,161.18 99.966,161.486 99.659,161.794 C 99.25,162.1 99.046,162.303 98.74,162.303 C 98.025,162.61 97.003,162.201 95.675,161.179 C 94.347,160.157 93.633,159.443 93.427,158.728 C 93.324,158.42 94.039,156.684 95.368,153.518 C 96.798,150.453 98.024,147.694 99.045,145.243 C 99.965,142.996 101.089,140.134 102.622,136.763 C 104.052,133.494 104.767,131.655 104.665,131.349 C 104.665,131.349 104.665,131.45 104.767,131.553 C 104.869,131.553 104.767,131.655 104.561,131.655 C 101.498,132.677 98.432,133.494 95.674,134.312 C 95.162,134.414 94.959,134.414 94.856,134.414 C 94.753,134.516 94.55,134.619 94.244,134.721 C 93.937,134.925 93.733,135.028 93.529,135.129 C 92.712,135.334 91.997,135.028 91.179,134.211 C 90.361,133.393 89.851,132.678 89.543,131.963 C 89.442,131.554 89.442,131.146 89.442,130.737 C 89.543,130.328 89.851,130.125 90.157,129.818 C 90.77,129.613 91.893,129.103 93.631,128.489 C 95.265,127.877 97.001,127.263 98.533,126.753 C 101.598,125.833 104.664,124.915 107.831,124.097 C 108.24,124.097 108.546,123.994 108.648,123.891 C 108.955,123.891 109.261,123.688 109.669,123.279 C 110.078,122.973 110.383,122.87 110.587,122.768 C 110.792,122.666 111.302,122.768 112.12,122.871 C 112.937,123.076 113.448,123.177 113.653,123.28 C 114.062,123.383 114.368,123.689 114.777,124.098 C 115.082,124.404 115.186,124.608 115.288,124.608 L 115.288,124.608 C 115.288,124.711 115.491,124.711 115.798,124.813 C 116.004,124.813 116.207,124.916 116.31,125.12 C 116.413,125.325 115.798,126.755 114.674,129.412 C 113.55,132.169 112.121,135.643 110.384,139.73 C 110.179,140.139 109.975,140.547 109.67,140.956 C 106.502,148.823 104.969,152.807 104.969,153.012 C 104.969,152.807 105.583,152.603 106.808,152.195 C 107.014,152.195 107.116,152.092 107.116,152.092 C 109.158,151.582 111.099,151.071 112.939,150.662 C 114.779,150.253 115.698,150.05 115.596,150.05 C 116.822,149.744 117.946,149.744 118.763,150.152 C 119.681,150.663 120.294,151.071 120.498,151.684 C 120.703,152.604 120.805,153.319 120.805,153.932 C 120.805,154.546 120.498,154.954 119.784,155.158 C 119.578,155.261 119.375,155.363 119.171,155.363 C 118.863,155.465 118.66,155.465 118.556,155.567 C 118.251,155.567 117.842,155.772 117.127,156.078 C 116.31,156.385 115.697,156.588 115.186,156.793 C 114.369,156.997 112.938,157.405 111.099,157.917 C 109.158,158.427 107.625,158.835 106.501,159.143 C 105.889,159.347 104.968,159.653 103.948,160.062 C 102.824,160.471 102.109,160.674 101.7,160.777 C 101.499,160.771 101.191,160.874 100.885,160.976 L 100.885,160.976 z " **style="fill:#a16121;fill-rule:evenodd"** id="path7" /> *<!-- SVG continues on...-->*

Believe it or not, that's just one <path> element above, with a series of line and curve instructions for just one of the shapes that compose the cat (one of the Zs above its head). (The first line is a <g>element, which is just used for grouping purpose.) Note that instead of styling the interior color in afill attribute, a style attribute containing CSS was used, which is an equally acceptable alternative. The style fill-rule:evenodd designates which portion of the path should be considered the "interior"; for more details, see the ["Fill Properties" section](http://www.w3.org/TR/SVG/painting.html" \l "FillProperties" \t "_top) of the W3C SVG specification.

To revise this SVG graphic to make it "colorable" for our coloring book, all we need to do is remove the fill color from those <path>s that have them, so that just the outlines of the shapes remain; we'll also need a method of indicating which paths should be colorable, and which should not. To accomplish this, we'll strip all the fill properties from the <path> style tags, and replace them with a separate fill="white" attribute (as it will be helpful later on to have our fill-color information in a distinct attribute from the rest of the styling). We'll also add a new class attribute,class="colorable", which will distinguish those paths that should be able to be colored (e.g., full shapes like the triangles encircling the cat's face) from those that cannot (e.g., the cat's whiskers).

[Example 5-2](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#new_svg_markup) shows our new SVG snippet.

*Example 5-2. Snippet of new, colorless sleepy cat SVG XML*

**<g** id="g5"**>** **<path** d="M 100.885,160.976 C 100.375,161.18 99.966,161.486 99.659,161.794 C 99.25,162.1 99.046,162.303 98.74,162.303 C 98.025,162.61 97.003,162.201 95.675,161.179 C 94.347,160.157 93.633,159.443 93.427,158.728 C 93.324,158.42 94.039,156.684 95.368,153.518 C 96.798,150.453 98.024,147.694 99.045,145.243 C 99.965,142.996 101.089,140.134 102.622,136.763 C 104.052,133.494 104.767,131.655 104.665,131.349 C 104.665,131.349 104.665,131.45 104.767,131.553 C 104.869,131.553 104.767,131.655 104.561,131.655 C 101.498,132.677 98.432,133.494 95.674,134.312 C 95.162,134.414 94.959,134.414 94.856,134.414 C 94.753,134.516 94.55,134.619 94.244,134.721 C 93.937,134.925 93.733,135.028 93.529,135.129 C 92.712,135.334 91.997,135.028 91.179,134.211 C 90.361,133.393 89.851,132.678 89.543,131.963 C 89.442,131.554 89.442,131.146 89.442,130.737 C 89.543,130.328 89.851,130.125 90.157,129.818 C 90.77,129.613 91.893,129.103 93.631,128.489 C 95.265,127.877 97.001,127.263 98.533,126.753 C 101.598,125.833 104.664,124.915 107.831,124.097 C 108.24,124.097 108.546,123.994 108.648,123.891 C 108.955,123.891 109.261,123.688 109.669,123.279 C 110.078,122.973 110.383,122.87 110.587,122.768 C 110.792,122.666 111.302,122.768 112.12,122.871 C 112.937,123.076 113.448,123.177 113.653,123.28 C 114.062,123.383 114.368,123.689 114.777,124.098 C 115.082,124.404 115.186,124.608 115.288,124.608 L 115.288,124.608 C 115.288,124.711 115.491,124.711 115.798,124.813 C 116.004,124.813 116.207,124.916 116.31,125.12 C 116.413,125.325 115.798,126.755 114.674,129.412 C 113.55,132.169 112.121,135.643 110.384,139.73 C 110.179,140.139 109.975,140.547 109.67,140.956 C 106.502,148.823 104.969,152.807 104.969,153.012 C 104.969,152.807 105.583,152.603 106.808,152.195 C 107.014,152.195 107.116,152.092 107.116,152.092 C 109.158,151.582 111.099,151.071 112.939,150.662 C 114.779,150.253 115.698,150.05 115.596,150.05 C 116.822,149.744 117.946,149.744 118.763,150.152 C 119.681,150.663 120.294,151.071 120.498,151.684 C 120.703,152.604 120.805,153.319 120.805,153.932 C 120.805,154.546 120.498,154.954 119.784,155.158 C 119.578,155.261 119.375,155.363 119.171,155.363 C 118.863,155.465 118.66,155.465 118.556,155.567 C 118.251,155.567 117.842,155.772 117.127,156.078 C 116.31,156.385 115.697,156.588 115.186,156.793 C 114.369,156.997 112.938,157.405 111.099,157.917 C 109.158,158.427 107.625,158.835 106.501,159.143 C 105.889,159.347 104.968,159.653 103.948,160.062 C 102.824,160.471 102.109,160.674 101.7,160.777 C 101.499,160.771 101.191,160.874 100.885,160.976 L 100.885,160.976 z " **style="fill-rule:evenodd" fill="white" class="colorable"** id="path7" /> *<!-- SVG continues on...-->*

And [Figure 5-3](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#colorless_sleeping_cat_screen) shows our colorless sleeping cat, ready for insertion into the coloring book.



*Figure 5-3. Sleepy cat SVG image, now colorless*

Next, we'll construct our coloring-book interface in HTML and embed our colorless cat SVG within the page.

**Constructing the Coloring Book HTML**

For our coloring book page, we'll just need a few elements:

* Our sleepy-cat image
* A palette for choosing the colors to fill the picture sections
* A couple interface buttons: Undo/Redo and Reset Drawing

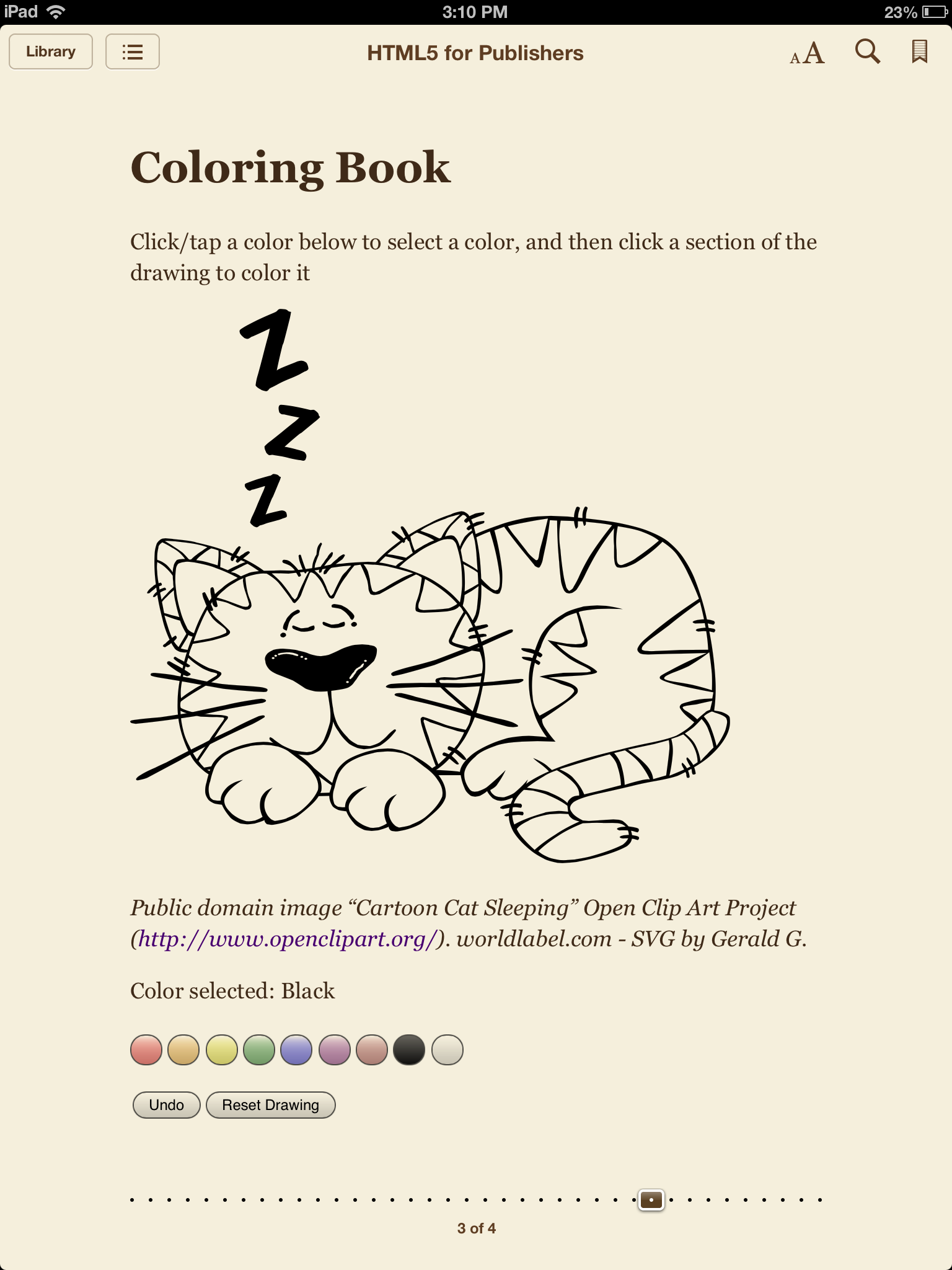
For our color palette, we can reuse the HTML we created in [“Canvas Finger Painting”](http://chimera.labs.oreilly.com/books/1234000000770/ch01.html#finger_painting_section), and we can add a new "Undo" button to go along with the "Reset Drawing" button. Then all we'll need to do is embed the SVG markup for the cat within the page. [Example 5-3](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#coloring_book_html) shows the HTML.

The full SVG content is redacted in order to save space. The full HTML code is available for download from the [HTML5 for Publishers GitHub repo](https://github.com/sandersk/HTML5-for-Publishers/blob/master/svg_coloring_book/coloring_book.html" \t "_top).

*Example 5-3. SVG Coloring Book HTML ([coloring\_book.html](https://github.com/sandersk/HTML5-for-Publishers/blob/master/svg_coloring_book/coloring_book.html" \t "_top))*

<?xml version="1.0" encoding="UTF-8" standalone="no"?> <!DOCTYPE html> <html xmlns="http://www.w3.org/1999/xhtml"> <head> <title>Coloring Book</title> <script src="jquery-1.6.2.min.js"></script> <script src="coloring\_book.js"></script> </head> <body> <h1>Coloring Book</h1> <p>Click/tap a color below to select a color, and then click a section of the drawing to color it</p> <div id="coloring\_book\_image"> <!-- Embedding SVG directly in HTML for maximum compatibility across browsers/ereaders --> <svg xmlns:svg="http://www.w3.org/2000/svg" xmlns="http://www.w3.org/2000/svg" version="1.1" style="width: 87%;" viewBox="0 0 446.379 412.104" id="Layer\_1" xml:space="preserve"> <!-- Full SVG redacted for space; see https://github.com/sandersk/HTML5-for-Publishers/blob/master/svg\_coloring\_book/coloring\_book.html for full code listing --> </svg> </div> <p><em>Public domain image &#x201c;Cartoon Cat Sleeping&#x201d; Open Clip Art Project (<a href="http://openclipart.org/people/Gerald\_G/Gerald\_G\_Cartoon\_Cat\_Sleeping.svg"> http://www.openclipart.org/</a>). worldlabel.com - SVG by Gerald G.</em></p> <p>Color selected: <span id="color\_chosen">Black</span></p> <p> <input type="button" class="color\_choice" id="Red" style="background-color: red; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Orange" style="background-color: orange; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Yellow" style="background-color: yellow; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Green" style="background-color: green; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Blue" style="background-color: blue; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Purple" style="background-color: purple; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Brown" style="background-color: brown; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="Black" style="background-color: black; width: 25px; height: 25px;"/> <input type="button" class="color\_choice" id="White" style="background-color: white; width: 25px; height: 25px;"/> </p> <p><input type="button" id="undo\_redo" value="Undo"/><input type="button" id="reset\_image" value="Reset Drawing"/></p> </body> </html>

[Figure 5-4](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#svg_coloring_book_ibooks_initial) shows the Coloring Book page in the iBooks reader for iPad.



*Figure 5-4. SVG Coloring Book page in iBooks for iPad*

**Embedding SVG in HTML**

In [Example 5-3](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#coloring_book_html), we embedded the full <svg> content directly in our HTML page to display the SVG graphic. However, there are several other methods for embedding SVG in HTML:

<img>*tag*

You can embed SVG graphics directly in HTML using an <img> element, just as you would embed a JPEG, PNG, or GIF graphic:

**<img** src="sleepy\_cat.svg"**/>**

The main downside of this approach is that when SVG is embedded with <img>, the SVG XML content is not accessible in the DOM, which means you cannot interact with the graphic via JavaScript. So the <img> method is not appropriate if you're looking to script the SVG.

<object>*tag*

Just as with <img>, you can embed SVG graphics directly in HTML using an <object> element:

**<object** data="sleepy\_cat.svg" type="image/svg+xml"**></object>**

However, the same caveats that apply to <img> apply to <object> as well. Scriptability of SVG is not feasible when using the <object> tag.

<embed>*tag*

The <embed> element is yet another option for SVG embedding:

**<embed** src="colorless\_cat.svg" type="image/svg+xml"**/>**

The good news is that SVG elements are accessible via the DOM when you embed with<embed>. But <embed> also comes with a caveat of its own. First, this element is new to HTML5, and is not supported in HTML4 environments. As such, it may not be compatible in all Web browsers and ereaders. Most notably, based on my personal testing, it's not currently compatible with the iBooks reader for iPhone/iPod/iPad, so if you're looking to develop interactive SVG content for iBooks, <embed> won't meet your needs.

<iframe>*tag*

Need a fourth option for embedding SVG? You can try the <iframe> element:

**<iframe** src="colorless\_cat.svg"**></iframe>**

Like <embed>, <iframe> SVG content is accessible via the DOM, so you can script it with JavaScript. So what are the downsides of this approach, you ask? Well, <iframe> was designed for embedding another document within the HTML document *in a frame*, which means that by default, most browsers will draw a border around the content. So if you want to suppress the border, you'll need to add extra CSS. Also, based on my personal testing, <iframe>d SVG content is not currently compatible with the iBooks reader.

*Skip the tagging and embed with Ajax calls*

If none of the above tagging methods meet your needs, you can also write JavaScript that will make an Ajax call to dynamically retrieve SVG content from an external file and embed it in the HTML document when the page is loaded. Here's what such a call looks like using jQuery:

$.ajax({ type: 'GET', url: 'cartoon\_sleeping\_cat.svg', dataType: 'html', success: function (svg\_resp, xmlstatus) { $('#coloring\_book\_image').append(svg\_resp); add\_coloring\_book\_events(); }, error: function (xhr, status, error) { alert(error); } });

Once the Ajax call has run, the SVG content will be embedded in the HTML document exactly as if you had hardcoded all the <svg> in the *.html* file yourself. This method should work in any Web browser with modern JavaScript support. Downside of this approach: based on my testing, the iBooks reader doesn't support Ajax calls to external documents, even if they are for files that local to the EPUB and not on the Internet.

So if you're developing EPUB 3 documents with scripted SVG files, and you want them to be supported in the iBooks reader, your best and only bet is to fall back on embedding all the <svg> right in your HTML document—hence, the approach in [Example 5-3](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#coloring_book_html).

For more details on the tagging approaches above, see this clear and concise [writeup from w3schools.com](http://www.w3schools.com/svg/svg_inhtml.asp" \t "_top).

**Constructing the Coloring Book JavaScript**

As with our [MathML equation solver](http://chimera.labs.oreilly.com/books/1234000000770/ch04.html#equation_solver_js), we'll use jQuery to simplify our code for scripting our sleepy-cat SVG to make it a "colorable" image. Four key pieces of functionality we need to write:

1. Code that lets the user click/tap a "colorable" section of the SVG graphic and fill it with the currently selected color
2. Code to let the user select a new color from the palette below the drawing
3. Code for the Undo/Redo button to let the user "undo" or "redo" their previous color action
4. Code for the Reset Drawing button to "reset" the entire drawing, reverting the color of every section back to white.

[Example 5-4](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#coloring_book_js_code) shows the complete coloring book JavaScript, with each of the above steps annotated with an explication of the code.

*Example 5-4. SVG Coloring Book JavaScript ([coloring\_book.js](https://github.com/sandersk/HTML5-for-Publishers/blob/master/svg_coloring_book/coloring_book.js" \t "_top))*

window.addEventListener('load', eventWindowLoaded, false); [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/1.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23scallout1) var undo\_element = $('path[class="colorable"]')[0]; var undo\_to\_color = "white"; function eventWindowLoaded() { add\_coloring\_book\_events(); } function add\_coloring\_book\_events() { *// Add click events for colorable portions of drawing* *// Oddly, the selector $('path.colorable') does not work in iBooks reader,*  *// although it does in Mobile Safari* $('path[class="colorable"]').bind("click", function(event) { [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/2.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23scallout2) *// Suppress default; helpful on touchscreen devices* event.preventDefault(); *// Get the current element and color and save it in undo\_element*  *// and undo\_to\_color variables* undo\_element = this; undo\_to\_color = $(this).attr("fill"); *// Toggle the "Undo" button to make sure it says "Undo" (it might say "Redo")* $('#undo\_redo').attr("value", "Undo"); *// Set the fill of clicked portion of drawing to the color chosen*  *// in palette below* color\_chosen = $("#color\_chosen").html(); $(this).attr("fill", color\_chosen); }); *// Add click events for color palette* $('.color\_choice').bind("click", function(event) { [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/3.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23scallout3) *// Get button id, which is the color name*  color\_chosen = $(this).attr("id"); *// Set color\_chosen text to the name of color clicked* $("#color\_chosen").html(color\_chosen); }); *// Add click events for reset button, which reverts the fill of the entire*  *// drawing to white* $('#reset\_image').bind("click", function(event) { [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/4.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23scallout4) *// Get all the colorable elements and set fill back to white* $('path[class="colorable"]').attr("fill", "white"); *// Resetting the drawing clears all undo information* $('#undo\_redo').attr("value", "Undo"); undo\_element = $('path[class="colorable"]')[0]; undo\_to\_color = "white"; }); $('#undo\_redo').bind("click", function(event) { [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/5.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23scallout5) *// First, save the existing color of the element we're going to undo* existing\_color = $(undo\_element).attr("fill"); *// Now revert the color back to the undo\_to\_color* $(undo\_element).attr("fill", undo\_to\_color); *// Finally, make existing\_color the new undo\_to\_color,*  *// to support "Redo" functionality* undo\_to\_color = existing\_color; *// If the button is named "Undo", rename it "Redo" and vice versa* if ($(this).attr("value") == "Undo") { $(this).attr("value", "Redo"); } else { $(this).attr("value", "Undo"); } }); }

|  |  |
| --- | --- |
| [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/1.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23sco1) | The first two blocks are setup code. We set up an event listener to trigger theadd\_coloring\_book\_events() function when the page has finished loading, and we set default values for the undo\_element and undo\_to\_color variables, which will later be used to track "undo" data as the user starts coloring in the cat. |
| [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/2.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23sco2) | In this block, we bind a click event[[8]](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#ftn.id566198) (see [“Scripting the Equation Solver with JavaScript”](http://chimera.labs.oreilly.com/books/1234000000770/ch04.html#equation_solver_js) for more on event binding with jQuery) to all <path> elements in the SVG that have a class value ofcolorable (via the selector $('path[class="colorable"]')). We call the preventDefault()function to suppress any default browser behaviors that would otherwise be associated with the click (helpful in iBooks, specifically, where a tap would normally trigger display of the reader menu up top).  To support our Undo/Redo button, we store the element just clicked (this) in the undo\_elementvariable, and its current color in undo\_to\_color. Since the user triggered a "color" action by clicking on a fillable path in the image, we update the Undo/Redo button's text to say "Undo", so he will know he can undo this action if he desires.  With the "undo" data now stored, we're ready to color the clicked section of the cat with the current color. The current color is stored in the element with id color\_chosen, so we grab that color and then set the fill attribute on the clicked element (again, this) to this color. This will fill the interior of the <path> with the current color. |
| [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/3.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23sco3) | In this block, we bind click events to all the buttons in our color palette (which all have a class value of color\_choice). When a click is triggered on a button, we get the name of the color (which corresponds to the id on the button) and then update the color\_chosen element text with the color name, so that the user will see which color is currently selected, and the function bound to the <path> elements can query the color\_chosen field to get the current color. |
| [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/4.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23sco4) | In this block, we bind a click event to the Reset Drawing button (with id of reset\_image). When Reset Drawing is pressed, we want to revert the fill of all colorable <path>s in the drawing back to white, so we use the selector $('path[class="colorable"]') to get those elements and then call attr() on them to set their fill attributes back to white. When we reset the drawing, for simplicity, we'll also purge all existing "undo" information and revert theundo\_element and undo\_to\_color variables back to their default values. |
| [http://orm-chimera-prod.s3.amazonaws.com/assets/callouts/5.png](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html%23sco5) | This last block binds a click event to the Undo/Redo button (with id of undo\_redo). The desired functionality of the Undo/Redo button is as follows:   * The button starts out being labeled as "Undo". When the user clicks the Undo button after coloring a section of the cat, we want to revert the color of that section back to the color it was prior to being clicked. (e.g., if the cat's face was white, and then the user colored it yellow, clicking Undo should revert it back to white). Then we want to change the label on the button to Redo, to give the user an opportunity to override his last Undo (e.g., switch the face back to give it a yellow fill again) * When the button is labeled "Redo", we want to swap the color of the clicked section back to its color prior to clicking Undo (i.e., redoing the change that was just undone). After Redo is clicked, we want to switch the button back to being labeled Undo again, in case the user wants to Undo the change again; effectively, he can toggle back and forth between the current and previous states by clicking Undo and then Redo in succession.   As discussed previously, when the user clicks a new section in the image (whether after pressing Undo/Redo or not), we switch the label on the button back to "Undo" and update the "undo" information stored in undo\_element and undo\_to\_color. This effectively means we're storing just one level of "undo" information for our image, meaning the user can only undo his most recent coloring action.  To accomplish the above specs, the first thing we do is store the existing color of the undo\_element in existing\_color. Next we revert the undo\_element’s fill color back to theundo\_to\_color. Then, to support "redo" functionality, we store the existing\_color variable as undo\_to\_color, which effectively makes the color prior to the "undo" the new "undo color." Lastly, if the button is currently labeled "Undo", we change it to say "Redo", and vice versa. If the user were to click Undo/Redo again in succession, the fill color and undo\_to\_color will also be flip-flopped again, reversing their previous Undo/Redo action. |

[Figure 5-5](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#svg_coloring_book_image_complete) shows a colored-in sleepy cat in iBooks for iPad. You can download the full example code from the *[HTML5 for Publishers](https://github.com/sandersk/HTML5-for-Publishers/tree/master/svg_coloring_book" \t "_top)*[GitHub repo](https://github.com/sandersk/HTML5-for-Publishers/tree/master/svg_coloring_book" \t "_top), and [try out the example for yourself](http://examples.oreilly.com/0636920022473/svg_coloring_book/coloring_book.html" \t "_top) in your browser or ereader



*Figure 5-5. SVG Coloring Book completed "sleepy cat" image; he's masquerading as a tiger!*

**Try It Now!**

Try out the SVG coloring book below:

**SVG, EPUB, and Ereader Compatibility**

While SVG rendering is supported in all major modern desktop browsers (including Internet Explorer, Safari, Firefox, and Google Chrome), support in the major ereaders is a little more spotty. Per the [latest data compiled by the Book Industry Study Group (BISG)](http://www.bisg.org/what-we-do-12-152-epub-30-support-grid.php" \t "_top), as of January 2013, SVG image rendering is supported in iBooks, Adobe Digital Editions, Kinde Fire, the latest Kindle eInk readers that support the KF8 format, NOOK tablet and NOOK apps (but not NOOK Simple Touch), and the Google Play app. However, it's important to make a distinction between the capability to simply render SVG (as the aforementioned readers can) and the ability to both *render and script SVG via JavaScript*. Currently, iBooks is the only major ereading platform that supports SVG scriptability via JavaScript, but hopefully this is something that will change with more widespread EPUB 3 adoption in the future.

**SVG Versus Canvas: Which Should You Use?**

Both SVG graphics and Canvas provide mechanisms for doing interactive image content in HTML5, so which should you use in your projects? To a certain extent, it's just a matter of preference, and whether you prefer manipulating SVG XML via JavaScript or prefer drawing natively with the Canvas API. However, there are a few differences between the two methods that may further guide your decision:

*Canvas is "stateless"; SVG is "stateful"*

When you do a series of drawing operations on a <canvas> element, the only information that's maintained by the Canvas API is the current state of the image after the most recent drawing operation. If you draw a circle on the Canvas and then draw a larger square that covers it completely, the Canvas does not "remember" that there is a circle below it. So if you wanted to move that same circle so that it were on top of the square instead of below it, the easiest approach would probably be to wipe the whole <canvas> clean, redraw the square, and then redraw the circle on top of it.

By contrast, in SVG, the data for each element in your image is stored in a corresponding element within your main <svg> tag, which means that there's a record of every drawing instruction that composes your graphic. So if you drew a circle and then a square on top of it, and wanted to move the circle or the square, you could just simply update the data in the corresponding <circle> or <rect> element. Depending on how you're manipulating your images, this "statefulness" could be a huge benefit.

*Canvas is bitmap; SVG is vector*

When you're drawing on a <canvas> element, you're working in a bitmap, pixel-based environment. Image data is represented as a series of pixels, and if you want, you can actually compose graphics by scripting the creation and placement of pixels, one by one. For more details on this technique, see the article ["pushing pixels with canvas"](http://hacks.mozilla.org/2009/06/pushing-pixels-with-canvas/" \t "_top) from the Mozilla Hacks blog.

In SVG, as we've discussed in this chapter, your graphics are vector-based and are composed of line/curve/shape operation data, not individual pixel data. If your needs are well served by manipulating pixels, Canvas is likely a better choice; conversely, if you're better served manipulating vectors, you'll probably prefer working with SVG.

*Canvas can manipulate HTML5 video*

If you're interested in doing work with HTML5 video—for example, taking frames from an MP4 video embedded in a <video> on the page and drawing text captions or images on top of them—you'll probably want to work with Canvas, which can integrate video content quite easily. For more details on techinques for adding image content from a video into Canvas, check out Chapter 6, "Mixing HTML5 Video and Canvas" from *[HTML5 Canvas](http://bitly.com/ormhtml5canvas" \t "_top)* (O'Reilly).

**Bibliography/Additional SVG Resources**

Here are some additional recommended resources if you are interested in learning more about the SVG topics discussed in this chapter:

*[W3C "Scalable Vector Graphics (SVG) 1.1" specification](http://www.w3.org/TR/SVG/" \t "_top)*

The definitive source of information about the SVG specification. A clear, well-written reference to exactly what's formally supported in the SVG specification, with descriptions of all SVG elements and their syntax, containing occasional examples as appropriate.

*[SVG Essentials](http://bit.ly/svgessentials" \t "_top) by J. David Eisenberg (O'Reilly Media)*

A comprehensive guide to the SVG standard, which discusses all SVG elements in detail, SVG scripting and animation, and methods for generating SVG content programmatically.

*["Elliptical Curves"](http://www.itk.ilstu.edu/faculty/javila/SVG/SVG_drawing1/elliptical_curve.htm" \t "_top)*

A brief but detailed tutorial from the site for Illinois State's School of Technology on drawing elliptical curves in SVG. If you're looking to learn more about drawing arcs in your SVG <path>s, check out this reference.

*["SVG In HTML Pages"](http://www.w3schools.com/svg/svg_inhtml.asp" \t "_top)*

Another concise, detailed tutorial on the various options for embedding SVG in HTML. Worth a read for anyone planning to script SVG.

*[SVG "Animation",](http://www.w3.org/TR/SVG/animate.html" \t "_top) from the W3C 1.1 Specification*

Worth calling out separately is the "Animation" section of the W3C SVG spec, which discusses SVG's native animation support (no JavaScript needed) via the <animate> element. <animate>was not covered in this chapter, but if you're interested in exploring the animation possibilities with SVG, definitely check out this portion of the spec.

*[Inkscape](http://inkscape.org/" \t "_top)*

An open source editor that specializes in SVG. Check out this application if you're looking to create your own SVG graphics.

[[8]](http://chimera.labs.oreilly.com/books/1234000000770/ch05.html#id566198)The click event corresponds to either a click of the mouse, or a tap on a touchscreen device. For conciseness, references to "click" in this section mean "click or tap".