

THE GUIDE OF HTML5 AND JAVA SCRIPT



PROGRAMMING FOR BEGINNERS

**STEP BY
STEP GUIDE TO
BUILDING YOUR
FIRST HTML AND
JAVA SCRIPT
PROJECTS**



BY :- JEANINE MEYER

The Guide Of

HTML5 AND JAVA SCRIPT

Programming for Beginners

By :- Jeanine Meyer

Illustrator by :- Mark Nunney

Page:- 678

HTML News

HTML Tutorial: Step by Step Guide to Building Your First HTML and Java Script Application

May 30, 2019

HTML 5 ~ This is a New Edition Book. By Which You will Become Proficient in HTML in step-by-step, We Have Made Our Books in such a way that it will Help you to Understand the New One. You can Easily and Freshly use your Home / Office.

Contents

Chapter 1: Building the HTML5 Logo – Drawing on Canvas, with Scaling, and Semantic Tags.

- [Introduction](#)
- [Project History and Critical Requirements](#)
- [HTML5, CSS, and JavaScript features](#)
- [Drawing paths on canvas](#)
- [Placing text on canvas and in the body of a document](#)
- [Coordinate transformations](#)
- [Using the range input element](#)
- [Building the application and making it your own](#)
- [Testing and uploading the application](#)
- [Summary](#)

Chapter 2: Family Collage: Manipulating Programmer-defined Objects on a Canvas

- [Introduction](#)
- [Critical Requirements](#)
- [HTML5, CSS, and JavaScript features](#)
- [JavaScript objects](#)
- [User interface](#)
- [Saving the canvas to an image](#)
- [Building the application and making it your own](#)
- [Testing and uploading the application](#)
- [Summary](#)

Chapter 3: Bouncing Video: Animating and Masking

HTML5 Video

- [Introduction](#)
- [Project History and Critical Requirements](#)
- [HTML5, CSS, and JavaScript Features](#)
- [Definition of the Body and the Window Dimensions](#)
- [Animation](#)
- [Video Drawn on Canvas and As a Movable Element](#)
- [Traveling Mask](#)
- [User Interface](#)
- [Building the Application and Making It Your Own](#)
- [Making the Application Your Own](#)
- [Testing and Uploading the Application](#)
- [Summary](#)

Chapter 4: Map Maker: Combining Google Maps and the Canvas

- [Introduction](#)
- [Latitude & Longitude and Other Critical Requirements](#)
- [HTML5, CSS, and JavaScript Features](#)
- [Google Maps API](#)
- [Canvas Graphics](#)
- [Cursor](#)
- [Events](#)
- [Calculating Distance and Rounding Values for Display](#)
- [Building the Application and Making It Your Own](#)
- [Testing and Uploading the Application](#)
- [Summary](#)

Chapter 5: Map Portal: Using Google Maps to Access Your Media

- [Introduction](#)
- [Project History and Critical Requirements](#)
- [HTML5, CSS, and JavaScript Features](#)
- [Google Maps API for Map Access and Event Handling](#)
- [Project Content](#)
- [Presentation and Removal of Video, Audio and Images](#)
- [Distances and Tolerances](#)
- [Regular Expressions](#)
- [External Script File](#)
- [Dynamic Creation of HTML5 Markup and Positioning](#)
- [Hint Button](#)
- [Shuffling](#)
- [Building the Application and Making It Your Own](#)
- [The Map videos Application](#)
- [The Map media base Application](#)
- [The Quiz Application](#)
- [Testing and Uploading the Application](#)
- [Summary](#)

Chapter 6: Where am I: Using Geolocation, the Google Maps API, and PHP

- [Introduction](#)
- [Geolocation and Other Critical Requirements](#)
- [HTML5, CSS, JavaScript, and PHP Features](#)
- [Geolocation](#)
- [Reverse Geocoding](#)
- [Clicking the Map](#)
- [Checking E-mail Address Input and Invoking PHP to send e-mail](#)
- [A Brief Introduction to the PHP Language](#)
- [Building the Application and Making It Your Own](#)
- [Testing and Uploading the Application](#)
- [Summary](#)

Who Is This Book For?

I do believe my explanations are complete, but I am not claiming, as I did for my previous book, *The Essential Guide to HTML5*, that this book is for the total beginner. This book is for the developer who has some knowledge of programming and who wants to build (more) substantial applications by combining basic features and combining JavaScript with other technologies. It also can serve as an idea book for someone working with programmers to get an understanding of what is possible.

How Is This Book Structured?

This book consists of ten chapters, each organized around an application or type of application. You can skip around. However, it probably makes sense to read Chapter 4 before 5 or 6. Also, the PHP server-side language is used in a simple way in Chapter 6 and then more fully in Chapter 10. Other cross-references are indicated in the text. Each chapter starts with an introduction to the application, with screenshots of the applications in use. In several cases, the differences between browsers are shown. The chapters continue with a discussion of the critical requirements, where concepts are introduced before diving into the technical details. The next sections describe how the requirements are satisfied, with specific constructs in HTML5, JavaScript, PHP, and/or SQL, and with standard programming techniques. I then show the application coding line by line with comments. Each chapter ends with instructions and tips for testing and uploading the application to a server, and a summary of what you learned. The code (with certain exceptions noted for Chapter 10) is all included as downloads available from the publisher. In addition, the figures are available as full-color TIFF files. Of course, you will want to use your own media for the projects shown in Chapters 2, 3, 5, and 8. My media (video, audio, images) is included with the code and this includes images for the 50

states for the states game in Chapter 9. You can use the project as a model for a different part of the world or a puzzle based on an image or diagram. Let's get started.

C H A P T E R - 1

C H A P T E R - 1

Building the HTML5 Logo – Drawing on Canvas, with Scaling, and Semantic Tags

In this chapter, we will review

- Drawing paths on a canvas
- Placing text on a canvas
- Coordinate transformations
- Fonts for text drawn on canvas and fonts for text in other elements
- Semantic tags
- The range input element

Introduction :

The project for this chapter is a presentation of the official HTML5 logo, with accompanying text. The shield and letters of the logo are drawn on a canvas element and the accompanying text demonstrates the use of semantic tags. The viewer can change the size of the logo using a slider input device. It is an appropriate start to this book, a collection of projects making use of HTML5, JavaScript and other technologies, because of the subject matter and because it serves as a good review of basic event-driven programming and other important features in HTML5. The way I developed the project, building on the work of others, is typical of how most of us work. In particular, the circumstances provide motivation for the use of coordinate transformations. Lastly, at the time of writing, Firefox does not fully implement the slider input element. Unfortunately, this also is a common situation and I will discuss the implications. The approach of this book is to explain HTML5, Cascading Style Sheets and JavaScript chapters in the context of specific examples. The projects represent a variety of applications and, hopefully, you will find something in each one that you will learn and adapt for your own purposes.

- Note If you need an introduction to programming using HTML5 and JavaScript, you can consult my book the Essential Guide to HTML5 or other books published by Après or others. There also is considerable material available online

Note If you need an introduction to programming using HTML5 and JavaScript, you can consult my book the Essential Guide to HTML5 or other books published by

Après or others. There also is considerable material available online.

Figure 1-1 shows the opening screen for the logo project on the Chrome browser. (Skip ahead to Figure 1-3 for the appearance on Firefox.)



Scale percentage:



Note: slider treated as text field in some browsers.

Built on [work by Daniel Davis, et al.](#), but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

[HTML5 Logo by W3C.](#)

Figure 1-1. Opening Screen for HTML5 Logo

Notice the slider feature, the accompanying text, which contains what appears to be a hyperlink, and the text in a footer below a yellow line. The footer also includes a hyperlink. As I will explain later, the function and the formatting of the footer and any other semantic element is totally up to me, but providing a reference to the owners of the logo, The World Wide Web Consortium would be deemed an appropriate use.

The viewer can use the slider to change the size of the logo. Figure 1-2 shows the application after the slider has been adjusted to show the logo reduced to about a third in width and in height.



Scale percentage: Note: slider treated as text field in some browsers.
Built on [work by Daniel Davis, et al](#), but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

HTML5 Logo by [W3C](#).

Figure 1-2. Logo scaled down

The implementation of HTML5 is not complete by any browsers and, as it turns out, Firefox treats all slider inputs as simple text fields. This is termed 'graceful degradation' and it certainly is better than producing nothing at all.

Figure 1-3 shows the opening screen in Firefox. Notice the initial value is displayed as 100.

Figure 1-3. Application using Firefox

As will be the practice in each chapter, I now explain the critical requirements of the application, more or less independent of the fact that the implementation will be in HTML5, and then describe the features of HTML5, JavaScript, and other technologies as needed that will be used in the implementation. The Building section includes a table with comments for each line of code and also guidance for building similar applications. The Testing section provides details for uploading and testing. This section is more critical for some projects than others. Lastly, there is a Summary that reviews the programming concepts covered and previews what is next in the book.

Project History and Critical Requirements

The critical requirements for this project are somewhat artificial and not easily stated as something separate from HTML. For example, I wanted to draw the logo as opposed to copying an image from the Web. My design objectives always include wanting to practice programming and prepare examples for my students. The shape of the shield part of the logo seemed amenable to drawing on canvas and the HTML letters could be done using the draw text feature. In addition, there are practical advantages to drawing images instead of using image files. Separate files need to be managed, stored, and downloaded. The image shown in Figure 1-4 is 90KB. The file holding the code for the program is only 4KB. Drawing a logo or other graphic means that the scale and other attributes can be changed dynamically



using code.

Figure 1-4. Image of logo

I looked online and found an example of just the shield done by Daniel Davis, who works for Opera. This was great because it meant that I did not have to measure a copy of the logo image to get the coordinates. This begs the question of how he determined the coordinates. I don't know the answer, even though we had a pleasant exchange of emails. One possibility is to download the image and use the grid feature of image processing programs such as Adobe Photoshop or Corel Paint Shop Pro. Another possibility is to use (old-fashioned) transparent graph paper.

However, there was a problem with building on Daniel Davis's work. His application did not include the HTML letters. The solution to this was to position the letters on the screen and then move down so to speak to position the drawing of the shield using the coordinates provided in Daniel's example. The technical term for 'moving down the screen' is performing a coordinate transformation. So the ability to perform coordinate transformations became a critical requirement for this project.

I chose to write something about the logo and, in particular, give credit and references in the form of hyperlinks. I made the decision to reference the official source of the logo as brief text at the bottom of the document below a line. The reference to Daniel Davis was part of the writing in the body. We exchanged notes on font choices and I will discuss that more in the next

section.

In order to give the viewer something to do with the logo, I decided to present a means of changing the size. A good device for this is a slider with the minimum and maximum values and steps all specified. So the critical requirements for this application include drawing shapes and letters in a specific font, coordinate transformations, formatting a document with a main section and a footer section, and including hyperlinks.

HTML5, CSS, and JavaScript features

I assume that you, the reader, have some experience with HTML and HTML5 documents. One of the most important new features in HTML5 is the canvas element for drawing. I describe briefly the drawing of filled-in paths of the appropriate color and filled-in text. Next, I describe coordinate transformations, used in this project for the two parts of the logo itself and for scaling, changing the size, of the whole logo. Lastly, I describe the range input element. This produces the slider.

Drawing paths on canvas

Canvas is a type of element introduced in HTML5. All canvas elements have a property (aka attribute) called the 2D context. Typically, a variable is set to this property after the document is loaded:

```
ctx = document.getElementById('canvas').getContext('2d');
```

It is important to understand that canvas is a good name: code applies color to the pixels of the canvas, just like paint. Code written later can put a different color on the canvas. The old color does not show through. Even though our code causes rectangles and shapes and letters to appear, these distinct entities do not retain their identity as objects to be re-positioned.

The shield is produced by drawing six filled-in paths in succession with the accumulated results as shown in Figure 1-5. You can refer to this picture when examining the code. Keep in mind that in the coordinates, the first number is the distance from the left edge of the canvas and the second number is the distance from the top edge of the canvas.

Figure 1-5. Sequence of paths for drawing logo

By the way, I chose to show you the sequence with the accumulated results. If I displayed what is drawn, you would not see the white parts making up the left side of the five. You can see it because it is two white filled-in paths on top of the orange.

All drawing is done using methods and properties of the ctx variable holding the 2D context property of the canvas element. The color for any subsequent fill operation is set by assigning a color to the fillStyleproperty of the canvas context.

```
ctx.fillStyle = "#E34C26";
```

This particular color, given in the hexadecimal format, where the first two hexadecimal (base 16) digits represent red, the second two hexadecimal digits represent green and the last two represent blue, is provided by the W3C website, along with the other colors, as the particular orange for the background of the shield. It may be counterintuitive, but in this system, white is specified by the value #FFFFFF. Think of this as all colors together make white. The absence of color is black and specified by #000000. The pearly gray used for the right hand side of the 5 in the logo has the value #EBEBEB. This is a high value, close to white. It is not necessary that you memorize any of these values, but it is useful

to know black and white, and that a pure red is #FF0000, a pure green is #00FF00 and a pure blue #0000FF. You can use the eyedropper/color picker tool in drawing programs such as Adobe Photoshop, Corel Paint Shop Pro or the online tool: <http://pixlr.com/> to find out values of colors in images OR you can use the official designation, when available, for official images.

All drawing is done using the 2 dimensional coordinate systems. Shapes are produced using the path methods. These assume a current location, which you can think of as the position of a pen or paint brush over the canvas. The critical methods are moving to a location and setting up a line from the current location to the indicated location. The following set of statements draws the five sided orange shape starting at the lower, left corner. The closePath method closes up the path by drawing a line back to the starting point.

```
ctx.fillStyle = "#E34C26";
```

```
ctx.beginPath();
```

```
ctx.moveTo(39, 250);
```

```
ctx.lineTo(17, 0);
```

```
ctx.lineTo(262, 0);
```

```
ctx.lineTo(239, 250);
```

```
ctx.lineTo(139, 278);
```

```
ctx.closePath();
```

```
ctx.fill();
```

If you haven't done any drawing on canvas, here is the whole HTML script needed to produce the 5-sided shape. The **onLoad** attribute in the body tag causes the **init** function to be invoked when the document is loaded. The **init** function sets the **ctx** variable, sets the **fillStyle** property and then draws the path.

```
<!DOCTYPE html> <html>

<head>

<title>HTML 5 Logo</title>

<meta charset="UTF-8">

<script>

function init() {

ctx = document.getElementById('canvas').getContext('2d'); ctx.fillStyle
= "#E34C26";

ctx.beginPath();

ctx.moveTo(39, 250);

ctx.lineTo(17, 0);

ctx.lineTo(262, 0);

ctx.lineTo(239, 250);
```

```
ctx.lineTo(139, 278);

ctx.closePath();

ctx.fill();

}

</script>

</head>

<body onLoad="init();">

<canvas id="canvas" width="600" height="400">

Your browser does not support the canvas element .

</canvas>

</body>

</html>
```

Do practice and experiment with drawing on the canvas if you haven't done so before, but I will go on. The other shapes are produced in a similar manner. By the way, if you see a line down the middle of the shield, this is an optical illusion.

Placing text on canvas and in the body of a document

Text is drawn on the canvas using methods and attributes of the context. The text can be filled in, using the **fillText** method or drawn as an outline using the **strokeText** method. The color is whatever the current **fillStyle** property or **strokeStyle** property holds. Another property of the context is the **font**. This property can contain the size of the text and one or more fonts. The purpose of including more than one font is to provide options to the browser if the first font is unavailable on the computer running the browser. For this project, I use

```
var fontfamily = "65px 'Gill Sans Ultra Bold', sans-serif";
```

and in the init function

```
ctx.font = fontfamily;
```

This directs the browser to use the Gill Sans Ultra Bold font if it is available and if not, use whatever the default sans-serif font on the computer.

I could have put this all in one statement, but chose to make it a variable. You can decide if my choice of font was close enough to the official W3C logo.

Note There are at least two other approaches to take for this example. One possibility is to NOT use text but to draw the letters as filled-in paths. The other is to locate and acquire a font and place it on the server holding the HTML5 document and reference

it directly using @font-face.

With the font and color set, the methods for drawing text require a string and a position: x and y coordinates. The statement in this project to draw the letters is

```
ctx.fillText("HTML", 31,60);
```

Formatting text in the rest of the HTML document, that is, outside a canvas element, requires the same attention to fonts. In this project, I choose to make use of the semantic elements new to HTML5 and follow the practice of putting formatting in the style element. The body of my HTML script contains two article elements and one footer elements. One article holds the input element with a comment and the other article holds the rest of the explanation. The footer element contains the reference to W3C. Formatting and usage of these are up to the developer/programmer. This includes making sure the footer is the last thing in the document. If I placed the footer before one or both articles, it would no longer be displayed at the foot, that is, the bottom of the document. The style directives for this project are the following:

```
footer {display:block; border-top: 1px solid orange; margin: 10px;  
font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; font-weight:  
bold;}
```

```
article {display:block; font-family: Georgia, "Times New Roman", Times, serif; margin: 5px;}
```

The styles each set up all instances of these elements to be displayed as blocks. This puts a line break before and after. The footer has a border on the top, which produces the line above the text. Both styles specify a list of four fonts each. So the browser first sees if Trebuchet MS is available, then checks for Arial, then for Helvetica and then, if still unsuccessful, uses the system default sans-serif font for the footer element. Similarly, the browser checks for Georgia, then Times New roman, then Times and then, if unsuccessful, uses the standard serif font. This probably is overkill, but it is the secure way to operate. The footer text is displayed in bold and the articles each have a margin around them of 5 pixels.

Formatting, including fonts, is important. HTML5 provides many features for formatting and for separating formatting from structure and content. You do need to treat the text on the canvas differently than the text in the other elements.

Coordinate transformations

I have given my motivation for using coordinate transformations, specifically to keep using a set of coordinates. To review, a coordinate system is the way to specify positions on the canvas. Positions are specified as distances from an origin point. For the two-dimensional canvas, two coordinates are necessary: the first coordinate, governing the horizontal, often called the x and the second coordinate, governing the vertical, called the y. A pesky fact is that when drawing to screens the y axis is flipped so the vertical is measured from the top of the canvas. The horizontal is measured from the left. This means that the point (100,200) is further down the screen than the point (100,100).

In the logo project, I wrote code to display the letters HTML and then move the origin to draw the rest of the logo. An analogy would be that I know the location of my house from the center of my town and so I can give directions to the center of town and then give directions to my house. The situation in which I draw the letters in the logo and 'move down the screen' requires the translate transformation. The translation is done just in the vertical. The amount of the translation is stored in a variable I named offsety:

```
var offsety = 80; ...  
  
ctx.fillText("HTML", 31, 60);  
  
ctx.translate(0, offsety);
```

Since I decided to provide a way for the viewer to change the size of the logo, I made use of the scale transformation. Continuing the analogy of directions, this is equivalent to changing the units. You may give some directions in miles (or kilometers) and other directions in yards or feet or meters or, maybe, blocks. The scaling can be done separately for each dimension. In this application, there is a variable called factor value that is set by the function invoked when the input is changed. The statement

```
ctx.scale(factorvalue, factorvalue);
```

changes the units for both the horizontal and vertical direction.

HTML5 provides a way to save the current state of the coordinate system and restore what you have saved. This is important if you need your code to get back to a previous state. The saving and restoring is done using what is termed a stack: last in first out. Restoring the coordinate state is termed popping the stack and saving the coordinate state is pushing something onto the stack. My logo project does not use this in its full power, but it is something to remember to investigate if you are doing more complex applications. In the logo project, my code saves the original state when the document is first loaded. Then before drawing the logo, it restores what was saved and then saves it again so it is available the next time. The code at the start of the function dologo, which draws the logo, starts off as follows:

```
function dologo() { var offsety = 80 ;
```

```
ctx.restore();

ctx.save();

ctx.clearRect(0,0,600,400);

ctx.scale(factorvalue,factorvalue);

ctx.fillText("HTML", 31,60);

ctx.translate(0,offsety);

// 5 sided orange background ctx.fillStyle = "#E34C26";

ctx.beginPath();

ctx.moveTo(39, 250);

ctx.lineTo(17, 0);

ctx.lineTo(262, 0);

ctx.lineTo(239, 250);

ctx.lineTo(139, 278);

ctx.closePath();

ctx.fill();

// right hand, lighter orange part of the background ctx.fillStyle =
"#F06529";

ctx.beginPath();
```

```
ctx.moveTo(139, 257);
ctx.lineTo(220, 234);
ctx.lineTo(239, 20);
ctx.lineTo(139, 20);
ctx.closePath();
ctx.fill();
```

...

Note that the canvas is cleared (erased) of anything that was previously drawn.

Using the range input element

The input device, which I call a slider, is the new HTML5 input type range, and is placed in the body of the HTML document. Mine is placed inside an article element. The attributes of this type and other input elements provide ways of specifying the initial value, the minimum and maximum values, the smallest increment adjustment and the action to take if the viewer changes the slider. The code is

```
<input id="slide" type="range" min="0" max="100"  
value="100" onChange="changescale(this.value)" step="10"/>
```

The min, max, (initial) value, and the step can be set to whatever you like. Since I was using percentage and since I did not want the logo to get bigger than the initial value or deal with negative values, I used 0 and 100.

In the proper implementation of the slider, the viewer does not see the initial value or the maximum or minimum. My code uses the input as a percentage. The expression **this.value** is interpreted as the value attribute of THIS element, emphasis given in capitals to convey the switch to English! The term **this** has special meaning in JavaScript and several other programming languages. The **changescale** function takes the value, specified by the parameter given in the assignment to the **onChange** attribute, and uses it to set a global variable (a variable declared outside of any function so it persists and is

available to any function).

```
function changescale(val) {  
    factorvalue = val / 100;  
    dologo();  
}
```

It is part of the specification of HTML5 that the browsers will provide form validation that is, check that the conditions specified by attributes in the input elements are obeyed. This can be a significant productivity boost in terms of reducing the work programmers need to do and a performance boost since the checking probably would be faster when done by the browser. We will discuss it more in Chapter 10 on databases and php. In the HTML5 logo project, an advantage of the slider is that the viewer does not need to be concerned with values but merely moves the device. There is no way to input an illegal value. I do not want to disparage the Firefox browser, and, as I indicated, producing a text box is better than producing nothing, but, at least at the time of writing, it does not display a slider or do any checking. Figure 1-6 shows the results of entering a value of 200 in the input field.

HTML



Scale percentage: Note: slider treated as text field in some browsers.

Built on [work by Daniel Davis, et al](#), but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

[HTML5 Logo by W3C](#)

The canvas is of fixed width and height and drawing outside the canvas, which is what is done when the scaling is done to accept numbers and stretch them out to twice the original value, is ignored.

Building the application and making it your own

The project does one thing, draw the logo. A function, **dologo**, is defined for this purpose. Informally, the outline of the program is

1. *init: initialization*
2. *dologo: draw the logo starting with the HTML letters and then the shield*
3. *changescale: change the scale*

The function called and calling table shows the relationship of the functions.

The **dologo** function is invoked when the document is first loaded and then whenever the scale is changed.

Table 1-1. Functions in the HTML5 Logo project

Function	Invoked / Called By	Calls
init	invoked by action of the onLoad attribute in the body tag	dologo
dologo	init and changescale	
changescale	invoked by action of the onChange attribute in the <input type="range"...> tag	dologo

The coding for the **dologo** function puts together the techniques previously described. In particular, the code brings back the original coordinate system and clears off the canvas.

The global variables in this application are

```
var ctx;  
  
var factorvalue = 1;  
  
var fontfamily = "65px 'Gill Sans Ultra Bold', sans-serif";
```

As indicated earlier, it would be possible to not use the **fontfamily** but use the string directly in the code. It is convenient to make **ctx** and **factorvalue** global.

Table 1-2 shows the code for the basic application, with comments for each line.

Table 1-2. Complete Code for the HTML5 Logo project

Code Line	Description
<!DOCTYPE html>	header
<html>	opening html tag
<head>	opening head tag
<title>HTML5 Logo </title>	complete title element
<meta charset="UTF-8">	meta tag
<style>	opening style tag
footer {display:block; border-top: 1px solid orange; margin: 10px; font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; font-weight: bold;}	style for the footer, including the top border and font family
article {display:block; font-family: Georgia, "Times New Roman", Times, serif; margin: 5px;}	style for the 2 articles
</style>	close the style element

<code><script language="JavaScript"></code>	Opening script tag. Note: case doesn't matter for the JavaScript.
<code>var ctx;</code>	Variable to hold the context. Used in all drawing
<code>var factorvalue = 1;</code>	set initial value for scaling
<code>var fontfamily = "65px 'Gill Sans Ultra Bold', sans-serif";</code>	set the fonts for the text drawn on the canvas
<code>function init() {</code>	start of init function
<code>ctx document.getElementById('canvas').getContext('2d');</code>	= set ctx
<code>ctx.font = fontfamily;</code>	set font for text drawn on canvas
<code>ctx.save();</code>	save the original coordinate state
<code>dologo();</code>	invoke function to draw the logo

Code Line	Description

<code>}</code>	close function
<code>function dologo() {</code>	start of dologo function
<code>var offsety = 80 ;</code>	specify amount to adjust the coordinates to draw the shield part of the logo.
<code>ctx.restore();</code>	restore original state of coordinates
<code>ctx.save();</code>	save it (push onto stack) so it can be restored again
<code>ctx.clearRect(0,0,600,400);</code>	erase the whole canvas
<code>ctx.scale(factorvalue,factorvalue);</code>	scale horizontally and vertically using value set by slider
<code>ctx.fillText("HTML", 31,60);</code>	draw the letters: HTML
<code>ctx.translate(0,offsety);</code>	move down the screen (canvas)
<code>// 5 sided orange background</code>	
<code>ctx.fillStyle = "#E34C26";</code>	set to official bright orange
<code>ctx.beginPath();</code>	start a path

<code>ctx.moveTo(39, 250);</code>	move to indicated position at lower left
<code>ctx.lineTo(17, 0);</code>	draw line up and more to the left
<code>ctx.lineTo(262, 0);</code>	draw line straight over to the right
<code>ctx.lineTo(239, 250);</code>	draw line down and slightly to the left
<code>ctx.lineTo(139, 278);</code>	draw line to the middle, low point of the shield
<code>ctx.closePath();</code>	close the path

Code Line	Description
<code>ctx.fill();</code>	fill in with the indicated color
<code>// right hand, lighter orange part of the // background</code>	
<code>ctx.fillStyle = "#F06529";</code>	set color to the official darker orange
<code>ctx.beginPath();</code>	start the path
<code>ctx.moveTo(139, 257);</code>	move to middle point, close to the top

<code>ctx.lineTo(220, 234);</code>	draw line to the right and slightly up
<code>ctx.lineTo(239, 20);</code>	draw line to the right and up
<code>ctx.lineTo(139, 20);</code>	draw line to the left (point at the middle)
<code>ctx.closePath();</code>	close path
<code>ctx.fill();</code>	fill in with the indicated color
<code>//light gray, left hand side part of the //five</code>	
<code>ctx.fillStyle = "#EBEBEB";</code>	set color to gray
<code>ctx.beginPath();</code>	start path
<code>ctx.moveTo(139, 113);</code>	move to middle horizontally, midway vertically
<code>ctx.lineTo(98, 113);</code>	draw line to the left
<code>ctx.lineTo(96, 82);</code>	draw line up and slightly further left
<code>ctx.lineTo(139, 82);</code>	draw line to right
<code>ctx.lineTo(139, 51);</code>	draw line up
<code>ctx.lineTo(62, 51);</code>	draw line to the left

Code Line	Description

<code>ctx.lineTo(70, 144);</code>	draw line to the left and down
<code>ctx.lineTo(139, 144);</code>	draw line to the right
<code>ctx.closePath();</code>	close path
<code>ctx.fill();</code>	fill in with indicated color
<code>ctx.beginPath();</code>	start a new path
<code>ctx.moveTo(139, 193);</code>	move to middle point
<code>ctx.lineTo(105, 184);</code>	draw line to the left and up
<code>ctx.lineTo(103, 159);</code>	draw line slightly to the left and up
<code>ctx.lineTo(72, 159);</code>	draw line more to the left
<code>ctx.lineTo(76, 207);</code>	draw line slightly to the right and down
<code>ctx.lineTo(139, 225);</code>	draw line to the left and down
<code>ctx.closePath();</code>	close path
<code>ctx.fill();</code>	fill in the shape in the indicated color
<code>// white, right hand side of the 5</code>	
<code>ctx.fillStyle = "#FFFFFF";</code>	set color to white
<code>ctx.beginPath();</code>	start path

<code>ctx.moveTo(139, 113);</code>	start at middle point
<code>ctx.lineTo(139, 144);</code>	draw line down
<code>ctx.lineTo(177, 144);</code>	draw line to the right
<code>ctx.lineTo(173, 184);</code>	draw line slightly left and down

Code Line	Description
<code>ctx.lineTo(139, 193);</code>	draw line more left and down
<code>ctx.lineTo(139, 225);</code>	draw line down
<code>ctx.lineTo(202, 207);</code>	draw line to the right and up
<code>ctx.lineTo(210, 113);</code>	draw line slightly right and up
<code>ctx.closePath();</code>	close path
<code>ctx.fill();</code>	fill in white
<code>ctx.beginPath();</code>	start a new path
<code>ctx.moveTo(139, 51);</code>	move to middle point
<code>ctx.lineTo(139, 82);</code>	move down
<code>ctx.lineTo(213, 82);</code>	move to the right
<code>ctx.lineTo(216, 51);</code>	move slightly to the right and up
<code>ctx.closePath();</code>	close path

<code>ctx.fill();</code>	fill in white
<code>}</code>	close dologo function
<code>function changescale(val) {</code>	open function changevalue with parameter
<code> factorvalue = val / 100;</code>	set factorvalue to the input divided by 100
<code> dologo();</code>	invoke function to draw logo
<code>}</code>	close changevalue function
<code></script></code>	close script element
<code></head></code>	close head element

Code Line	Description
<code><body onLoad="init();"></code>	body tag with attribute set to invoke init

<pre><canvas id="canvas" width="600" height="400"></pre>	<p>canvas tag setting dimensions and with id to be used in code</p>
<p>Your browser does not support the canvas element.</p>	<p>message to appear if canvas not supported</p>
<pre></canvas></pre>	<p>close canvas tag</p>
<pre><article></pre>	<p>article tag</p>
<pre>Scale percentage: <input id="slide" type="range" min="0" max="100" value="100" onChange="changescale(this.value)" step="10"/></pre>	<p>the slider (range) input with settings</p>
	<p>Comment to note</p>

<p>Note: slider treated as text field in some browsers.</p>	<p>that slider may be <u>text</u> <u>field</u>. It is still usable.</p>
<p></article></p>	<p>article close tag</p>
<p><article>Built on work by Daniel Davis, et al, but don't blame them for the fonts. Check out the use of font-family<u>> in the style element and the fontfamily variable in the script element for safe ways to do fonts. I did the scaling.</u> Note also use of semantic elements.</article></p>	<p>article tag with some text, including hyperlink</p>
<p><footer>HTML5 Logo by <abbr title="World Wide Web Consortium">W3C</abbr>.</p>	<p>footer tag and footer content, including abbr element</p>
<p></footer></p>	<p>footer close</p>

	tag
</body>	body close
</html>	html close

You can make this application your own by using all or parts of it with your own work. You probably want to omit the comments about fonts.

Testing and uploading the application

This is a simple application to test and upload (and test) because it is a single file. I am told that the logo does display on iPhone4 and iPad2, but the slider is a text box in each case. I also tested it on Safari and Opera on a PC. You can skip ahead to Chapter 8 for a project displaying a jigsaw puzzle turning into a video that does work with finger touches on the iPhone and iPad as well as mouse moves on computers.

Summary

In this chapter, you learned how make a specific drawing and also steps to take in producing other, similar, applications. The features used include

- paths
- text on the canvas and text in semantic elements in the body
- the range input element and its associated change event
- coordinate transformations, namely translate and scale
- specification of sets of fonts
- styles for semantic elements, including the border top to make a line to go before the footer

The next chapter describes how to build a utility application for making compositions or collages of photographs and shapes. It combines techniques of drawing on canvas and creating HTML elements with a standard technique in computing, objects. It also makes use of coordinate transformations.

Family Collage: Manipulating Programmer-defined Objects on a Canvas

In this chapter, you will learn about

- Creating and manipulating object-oriented programming for drawing on canvas
- Handling mouse events, including double-click
- saving the canvas to an image
- Using try and catch to trap errors
- Browser differences involving the location of the code
- Using algebra and geometry to construct shapes and determine when the cursor is over a specific object
- Controlling the icon used for the cursor

Introduction

The project for this chapter is a utility for manipulating objects on a canvas to produce a picture. I call it a utility because one person does the programming and gathers photographs and designs and then can offer the program to friends, family members, colleagues and others to produce the compositions / collages. The result can be anything from an abstract design to a collage of photographs. The objects in my example include two rectangles, an oval, a circle, a heart, and five family photographs. It is possible for you, or, perhaps, your end-user/customer/client/player, to make duplicate copies of any of the objects or to remove any of the objects. The end-user positions the object using drag and drop with the mouse. When the picture is judged to be complete, it is possible to create an image that can be downloaded into a file.

Figure 2-1 shows the opening screen for my program. Notice that you start off with ten objects to arrange.

Mouse down, move and mouse up to move objects. Double click for new object. Click re-load if images fail to appear.

[Open window with image \(which you can save into image file\)](#)

[Remove last object moved](#)

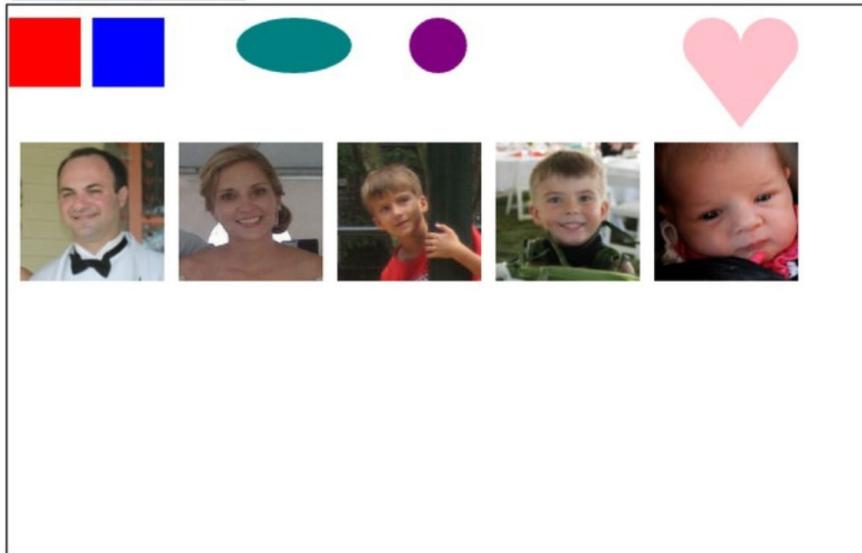


Figure 2-1. Opening screen for Family Pictures

Figure 2-2 shows what I, as an end-user, produced as a final product. I have arranged the photographs to represent my son's family, including my new granddaughter. I deleted the two rectangles and the oval and made duplicates of the circle to position as decoration in the corners of the picture.

Mouse down, move and mouse up to move objects. Double click for new object. Click re-load if images fail to appear.

[Open window with image \(which you can save into image file\)](#)

[Remove last object moved](#)

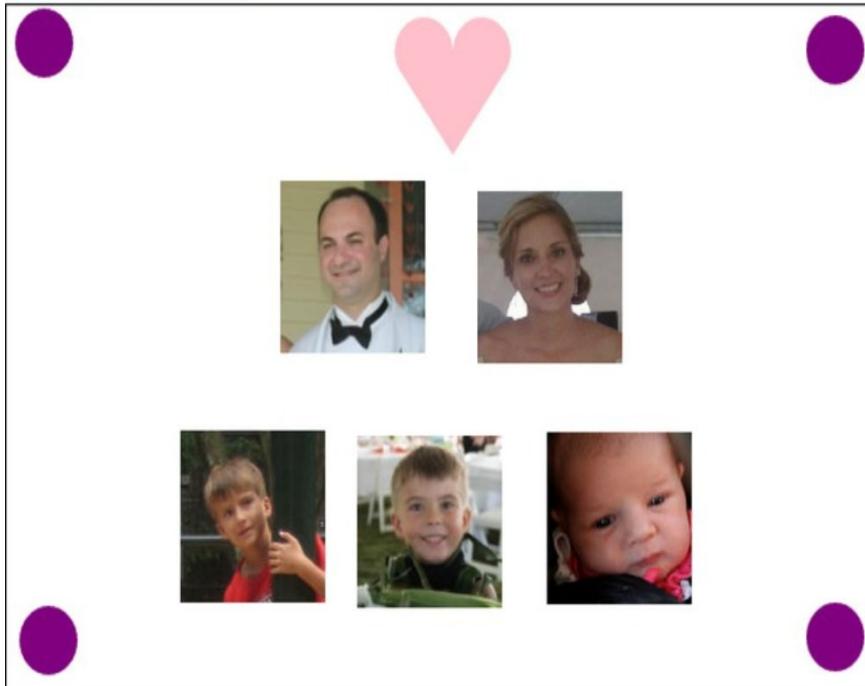


Figure 2-2. Sample final product: rearranged objects

I decided on including a heart, not just for sentimental reasons, but because it required me to use algebra and geometry. Don't be afraid of mathematics. It is very useful. I invented, so to speak, a canonical heart. For other shapes, you may be able to find a standard definition in terms of mathematical expressions.

In this situation, I created the set of objects and then I used the program to make a composition. You can plan your application to take family photographs or photographs categorized by another theme and some set of circles, ovals, rectangles, and hearts. When you are finished, you can offer this program to others for them to use. This is analogous to building a game program for players. The end-users for this application may be family members, friends, or colleagues.

Of course, it certainly is possible to use a drawing program such as Adobe Photoshop or Corel Paint Shop Pro to create compositions such as these, but this application provides considerable ease-of-use for its specific purpose. The project also serves as a vehicle to learn important programming techniques as

well as features of HTML5 and JavaScript. And, as will be a continual refrain, there are differences among the browsers to discuss.

Critical Requirements

The critical requirements for this project include constructing a framework for manipulating objects on the screen, including detecting mouse events on the objects, deleting objects and creating copies of objects. The current framework provides a way to specify rectangles, ovals, hearts and images, but the approach can accommodate other shapes and this is an important lesson of the chapter.

The objective is for the drag and drop operations to be reasonably precise: not merely moving something from one region of the window to another. I will revisit this topic in the Chapters 8 and 9 on making jigsaw puzzles.

I also made the decision to control the look of the cursor. The cursor when the mouse is not on the canvas is the standard arrow. When on the canvas element, the cursor will be the cross-hairs. When the user presses down on the mouse button and drags an object, the cursor changes to a hand with pointer finger.

When the work is complete, it is a natural desire to save it, perhaps as an image file, so this also is a requirement for the project.

HTML5, CSS, and JavaScript features

We now explore the features of HTML5 and JavaScript that are used for the Family Collage project. The idea is to maintain a list of the material on the canvas. This list will be a JavaScript array. The information will include the position of each item, how it is to be drawn on the canvas and how to determine if the mouse cursor is on the item.

JavaScript objects

Object oriented programming is a standard of computer science and a critical part of most programming languages. Objects have *attributes*, also called *properties*, and *methods*. A method is a function. Put another way, an object has data and code that may make use of the data. HTML and JavaScript have many built-in objects, such as **document** and **window** and also arrays and strings. For the Family Picture project, I make use of a basic facility in JavaScript (established before HTML5) for defining my own objects. These sometimes are called user-defined objects, but the term I and others prefer is programmer-defined objects. This is an important distinction for the Family Collage project in which you, the programmer, may create an application, with pictures and other shapes you identify and design, and then offer it to a family member to use.

The objective of this project is to set up a framework for creating and manipulating different shapes on the canvas, keeping in mind that once something is drawn to the canvas, its identity as a rectangle or image is lost. The first step for each shape is to define what is called a *constructor* function that stores the information that specifies the shape. The next step is to define the methods, code for using the information to do what needs to be done.

My approach gives the appearance of moving things on the canvas. In fact, information kept in internal variables is changed and the canvas is cleared and

new drawings made each time something happens to change the look of the canvas.

My strategy is to define new types of objects, each of which will have two methods defined

- **draw** for drawing the object on the canvas
- **over check** for determining if a given position, specifically the mouse position, is on the object

These methods reference the attributes of the object and use these values in mathematical expressions to produce the results. Once the constructor functions are defined, values can be created as new instances of these objects. An array called **stuff** holds all the object instances.

Note Object oriented programming in all its glory has a rich and often daunting vocabulary. Classes are what define objects. I have hinted here at what is called an interface. Classes can be subclasses of other classes and this may have been useful for pictures and rectangles. I'm aiming for a more casual tone here. For example, I will speak of objects and object instances.

Let's move away from generalities and show how this works. I created functions I named Rect, Oval, Picture and Heart. These will be what are called the

constructor functions for the Rect, Oval, Picture and Heart object instances. It is a convention to give such functions names starting with capital letters.

Rect

The definition of the **Rect constructor** function is

```
function Rect(x,y,w,h,c) {
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.w = w;
```

```
    this.h = h;
```

```
    this.draw = drawrect;
```

```
    this.color = c;
```

```
    this.overcheck = overrect;
```

```
}
```

The function is used as follows in the **init** function invoked when the document is loaded:

```
var r1 = new Rect(2,10,50,50,"red");
```

The variable r1 is declared and set to a new object constructed using the function **Rect**. The built-in term **new** does the task of creating a new object. The newly constructed object holds the values 2 and 10 for the initial x and y

position, accessed using the attribute names **x** and **y** and the values 50 and 50 for width and height accessed using the attribute names **w** and **h**. The term **this** refers to the object being constructed. The English meaning and the computer jargon meaning of *new* and *this* match. The **Rect** function also stores away values for the attributes draw and overcheck. It is not obvious from what you have seen so far, but these values will be used to invoke functions named **drawrect** and **overrect**.

This is the way to specify methods for the programmer-defined objects. Lastly, the **color** attribute is set to "red".

Oval

Moving on, the constructor function for **Oval** is similar.

```
function Oval(x,y,r,hor,ver,c)
{
    this.x = x;
    this.y = y;
    this.r = r;
    this.radsq = r*r;
    this.hor = hor;
    this.ver = ver;
```

```
this.draw = drawoval;  
  
this.color = c;  
  
this.overcheck = overoval;  
  
}
```

The **x** and **y** values refer to the center of the Oval. The **hor** and **ver** attributes will be used to scale the horizontal and vertical axis respectively and, depending on the values, produce an oval that is not a circle. The **radsq** attribute is calculated and stored to save time in the **overoval** function.

Note Computers are very fast and I am showing my age by storing away and then using the square of the radius. Still, making this trade-off of extra storage for savings in computation time may be justified.

In my example,

```
var oval1 = new Oval(200,30,20,2.0,1.0, "teal");
```

produces the teal colored oval in Figure 2-1. Notice that it is stretched out in the horizontal dimension. In contrast, the purple oval is declared in the following declaration:

```
var cir1 = new Oval(300,30,20,1.0,1.0,"purple");
```

The purple circle has the **hor** and **ver** values the same and so is a circle. You have every right to ask how or where this information is used to produce an oval or circle. The answer is in the **drawoval** function that will be shown later on. Similarly, the checking if a given x,y position is on the oval is performed by the **overoval** function.

Picture

The constructor for Picture objects stores away position, width and height, and the name of an Image object.

```
function Picture(x,y,w,h,imagename) {  
    this.x = x;  
    this.y = y;  
    this.w = w;  
    this.h = h;  
    this.imagename = imagename;  
    this.draw = drawpic;  
    this.overcheck = overrect;  
}
```

My example has four Picture objects. Here is code for setting up one of them:

```
var dad = new Image();  
  
dad.src = "daniel1.jpg";  
  
var pic1 = new Picture(10,100,100,100,dad);
```

Setting up one of the Picture objects takes more work than Rect objects. I need to write the code to create an **Image** object. This is a built-in object type in JavaScript. I need to acquire an image file and move it into the same folder as the HTML file and write code to assign the correct file name to the **src** attribute of the Image variable. (Alternatively, you can put all images into a subfolder or several subfolders. For example, the string for the **src** would be "**images/daniel1.jpg**".) Then I write the line of code that constructs the **pic1** variable.

Heart

We have one more of the programmer defined objects to cover. The challenge I set myself was to define values that specify a heart shape. I came up with the following: a heart shape is defined by the position, an x, y pair of values that will be the location of the cleft of the heart; the distance from the cleft to the bottom point, and the radius for the two partial circles representing the curved parts of the heart. You can think of this as a canonical heart. The critical pieces of information are shown in Figure 2-3. If and when you add new types of shapes to your application, you will need to invent or discover the data that defines the shape.

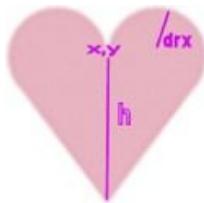


Figure 2-3. Data defining a heart

The constructor function saves the indicated values, along with the color, into

any newly constructed object. You might be suspecting that the drawing and the overcheck will be somewhat more complicated than the functions for rectangles and you would be correct. The constructor function resembles the other constructor function.

```
function Heart(x,y,h,drx,color)
  { this.x = x;
    this.y = y;
    this.h = h;
    this.drx = drx;
    this.radsq = drx*drx;
    this.color = color;
    this.draw = drawheart;
    this.overcheck = overheart;
    this.ang = .25*Math.PI;
  }
```

The **ang** attribute is a case of my hedging my bets. You notice that it is a constant and I could avoid making it an attribute. You will see later when I explain drawheart how my coding uses it to make the heart rounded. I made it an attribute just in case I want to change to allow hearts to have more

variability.

Drawing

I have the different method functions to explain, but let's go to where the drawing is done in order to demonstrate how all of this works together. I define an array, initially empty,

```
var stuff = [];
```

Then my code adds to this array with statements such as

```
stuff.push(pic1);
```

```
stuff.push(pic2);
```

```
stuff.push(pic3);
```

```
stuff.push(pic4);
```

```
stuff.push(r1);
```

```
stuff.push(s1);
```

```
stuff.push(oval1);
```

```
stuff.push(cir1);
```

At appropriate times, namely after any changes, the function **drawstuff** is invoked. It works by erasing the canvas, drawing a rectangle to make a frame, and then iterating over each element in the **stuff** array and invoking the **draw** methods. The function is

```
function drawstuff() {  
  
    ctx.clearRect(0,0,600,400);  
  
    ctx.strokeStyle = "black";  
  
    ctx.lineWidth = 2;  
  
    ctx.strokeRect(0,0,600,400);  
  
    for (var i=0;i<stuff.length;i++) {  
  
        stuff[i].draw();  
  
    }  
  
}
```

Notice that there is no coding that asks, is this an oval, if so do this, or is it a picture, if so do that.... Instead, the **draw** method that has been established for each member of the array does its work! The same magic happens when checking if a position (the mouse) is on an object. The benefit of this approach increases as more object types are added.

I did realize that since my code never changes the **strokeStyle** or the **lineWidth**, I could move those statements to the **init** function and just do them one time. However, it occurred to me that I might have a shape that does change these values and so to prepare for that possible change in the application at a later time, I would set **strokeStyle** and **lineWidth** in

drawstuff.

Now I will explain the methods for drawing and the methods for checking if a position is on the object. The **drawrect** function is pretty straight-forward:

```
function drawrect() {  
  
    ctx.fillStyle = this.color;  
  
    ctx.fillRect(this.x, this.y, this.w, this.h);  
  
}
```

Remember the term **this** refers to the object for which **drawrect** serves as a method. The **drawrect** function is the method for rectangles and for pictures.

The **drawoval** function is slightly, but only slightly, more complex. You need to recall how coordinate transformations work. HTML5 JavaScript only allows circular arcs but does allow scaling the coordinates to produce ovals (ellipses) that are not circles. What the coding in the **drawoval** function does is to save the current state of the coordinate system and then perform a translation to the center of the object.

Then a scaling transformation is applied, using the **hor** and **ver** properties. Now, after setting the **fillStyle** to be the color specified in the **color** attribute, I use the coding for drawing a path made up of a circular arc and

filling the path. The last step is to restore the original state of the coordinate system.

```
function drawoval() { ctx.save();

ctx.translate(this.x,this.y);

ctx.scale(this.hor,this.ver);

ctx.fillStyle = this.color;

ctx.beginPath();

ctx.arc(0,0,this.r,0,2*Math.PI,true);

ctx.closePath();

ctx.fill();

ctx.restore();

}
```

This is the way ovals that may or may not be circles are drawn on the canvas. Since my code restored the original state of the coordinate system, this has the effect of undoing the scaling and translation transformations.

The drawpic function is the easiest one, just one line:

Function	drawpic()	{
ctx.drawImage(this.imagename,this.x,this.y,this.w,this.h);		

}

Again, the terms starting with **this** followed by a dot and then the attribute names reference the stored attributes.

Note Please keep in mind that I didn't plan and program this whole application all at once. I did the rectangles and ovals and later added the pictures and much later the heart. I also added the duplication operation and the deletion operation much later. Working in stages is the way to go. Planning is important and useful, but you do not have to have all the details complete at the start.

The **drawheart** function starts by defining variables to be used later. The **leftctrx** is the x coordinate of the center of the left arc and the **rightctrx** is the x coordinate of the center of the right arc. The arcs are each more than a half circle. How much more? I decided to make this be $.25 * \text{Math.PI}$ and to store this value in the **ang** attribute.

The tricky thing was to determine where the arc stops on the right side. My code uses trig expressions to set the **cx** and **cy** values. The cx, cy position is where the arc meets the straight line. Figure 2-4 indicates the meaning of the variables.

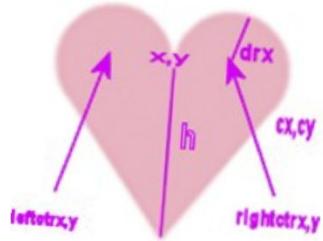


Figure 2-4. Added pieces of data used in functions

The path will start at what we are calling the cleft or the cleavage (giggle) and draw the arc on the

left, then draw a line to the bottom point, then up to the **cx,cy** point and then finish with the arc on the

right. The function is the following:

```
function drawheart() {  
  
    var leftctrx = this.x-this.drx;  
  
    var rightctrx = this.x+this.drx;  
  
    var cx = rightctrx+this.drx*Math.cos(this.ang);  
  
    var cy = this.y + this.drx*Math.sin(this.ang);
```

```
ctx.fillStyle = this.color;  
  
ctx.beginPath();  
  
ctx.moveTo(this.x,this.y);  
  
ctx.arc(leftctrx,this.y,this.drx,0,Math.PI-this.ang,true);  
  
ctx.arc(rightctrx,this.y,this.drx,this.ang,Math.PI,true);  
  
ctx.closePath();  
  
ctx.fill();  
  
}
```

Checking for mouse over object

Before describing the functions for the over check method, I will preview why it is needed. HTML5 and

There are three functions to explain for the **over check** method since Picture and Rect refer to the

same function. Each function takes two parameters. Think of the **mx, my** as the location of the mouse. The

overrect function checks for four conditions each being true. In English, the question is: Is **mx** greater

than or equal to **this.x** AND is **mx** less than or equal to **this.x + this.w** AND is **my** greater than or equal to

this.y AND is **my** less than or equal to **this.y + this.h**? The function says this more compactly:

```
function overrect (mx,my) {  
    if ((mx>=this.x)&&(mx<=(this.x+this.w))&&(my>=this.y)&&(my<=(this.y+this.h)))  
        {return true;}  
    else {return false;}}
```

}

The function defining the **over check** method for ovals is **over oval**. The over oval function performs the operation of checking if something is within a circle, but in a translated and scaled coordinate system. The check for a point being within a circle could be done by setting the center of the circle to x1,y1 and the point to x2,y2 and see if the distance between the two is less than the radius. I use a variation of this to save time and compare the square of the distance to the radius squared. I define a function **deists** that returns the square of the distance. But now I need to figure out how to do this in a translated and scaled coordinate system. The answer is to set x1,y1 to 0,0. This is the location of the center of the oval in the translated coordinate system. Then my code sets x2 and y2 as indicated in the code to what would be the scaled values.

```
function overoval(mx,my) { var x1 = 0;  
var y1 = 0;  
var x2 = (mx-this.x)/this.hor;  
var y2 = (my-this.y)/this.ver;  
if (distsq(x1,y1,x2,y2)<=(this.radsq) ){  
return true  
}
```

```
else {return false}  
}
```

This did not come to me instantly. I worked it out trying values for **mx** and **my** located in different positions relative to the oval center. The code does represent what the transformations do in terms of the translation and then the scaling.

The **overheart** function consists of several distinct **if** statements. This is a case of not trying for a simple expression but thinking about various situations. The function starts off by setting variables to be used later. The first check made by the function is to determine if the **mx,my** point is outside the rectangle that is the bounding rectangle for the heart. I wrote the outside function to return true if the position specified by the last two parameters was outside the rectangle indicated by the first four parameters. The **qx,qy** point is the upper left corner. **qwidth** is the width at the widest point and **qheight** is the total height. I thought of this as a quick check that would return false most of the time. The next two **if** statements determine if the **mx,my** point is contained in either circle. That is, I again use the comparison of the square of the distance from **mx,my** to the center of each arc to the stored **radsq** attribute. At this point in the function, that is, if the **mx,my** position was not close enough to the center of either circle and if **my** is above (less than) **this.y**, then the code returns false. Lastly, the code puts the **mx** value in the equation for

each of the sloping lines and compares the result to **my**. The equation for a line can be written using the slope **m** and a point on the line **x2,y2**:

$$y = m * (x - x2) + y2$$

The code sets **m** and **x2, y2** for the line on the left and then modifies it to work for the line on the right by changing the sign of **m**. One possible concern here is whether or not the fact that the screen coordinate system has upside down vertical values (vertical values increase going down the screen) causes a problem. I checked out cases and the code works.

```
function overheart(mx,my) {  
  
    var leftctrx = this.x-this.drx;  
  
    var rightctrx = this.x+this.drx;  
  
    var qx = this.x-2*this.drx;  
  
    var qy = this.y-this.drx;  
  
    var qwidth = 4*this.drx;  
  
    var qheight = this.drx+this.h;  
  
    //quick test if it is in bounding rectangle  
  
    if (outside(qx,qy,qwidth,qheight,mx,my)) {  
  
        return false;  
    }
```

```

//compare to two centers

if (distsq(mx,my,leftctrx,this.y)<this.radsq) return true;

if (distsq(mx,my,rightctrx,this.y)<this.radsq) return true;

// if outside of circles AND below (higher in the screen) than
this.y, return false if (my<this.y) return false;

// compare to each slope var x2 = this.x

var y2 = this.y + this.h;

var m= (this.h)/(2*this.drx); // left side

if (mx<=this.x) {

if (my < (m*(mx-x2)+y2)) {return true;}

else { return false; }

}

else {

m= -m;

if (my < (m*(mx-x2)+y2)) { return true}

else return false;

}

```

}

The reasoning underlying the **outside** function is similar to the overrect function. You need to write code comparing the **mx,my** value to the sides of the rectangle. However, for **outside** I chose to use the OR operator, **||**, and to return its value. This will be true if any of the factors are true and false otherwise.

```
function outside(x,y,w,h,mx,my) {  
    return ((mx < x) || (mx > (x+w)) || (my < y) || (my > (y+h)));  
}
```

Actually, what I said was true, but misses what could be an important consideration if performance is an issue. The **||** evaluates each of the conditions starting from the first (leftmost) one. As soon as one of them is true, it stops evaluating and returns true. The **&&** operator does a similar thing. As soon as one of the conditions is false, it returns false.

This is the basis for the four types of objects I designed for manipulation on the canvas. You can look ahead to examine all the code or continue to see how these objects are put in use in the responses to mouse events.

Note: This example does not demonstrate the full power of object-oriented

programming. In a language such as Java (or the variant Processing designed for artists), I could have programmed this in such a way to check that each additional object was defined properly, that is with the x and y attributes for location and methods for drawing and checking.

User interface

The application requirements for the user interface include dragging, that is, mouse down, mouse move and mouse up, for re-positioning items and double clicking for producing a duplicate copy of an item. I decided to use buttons for the other end-user actions: removing an item from the canvas and creating an image to be saved. The button action is straight-forward. I write two instances of the HTML5 button element with the **onClick** attributes set to the appropriate function.

```
<button onClick="saveasimage();">Open window with image (which  
you can save into image  
  
file)</button><br>  
<button onClick="removeobj();">Remove last object moved </button>
```

The **saveasimage** function will be explained in the next section. The **removeobj** function deletes the last moved object from the stuff array. The last moved object is the last one in the array. This makes the coding extremely simple:

```
function removeobj() {
```

```
stuff.pop();

drawstuff();

}
```

A **pop** for any array deletes the last element. The function then invokes the drawstuff function to display all but the last element. By the way, if the button is clicked at the start of the application, the last element pushed on the stuff array will be deleted. If this is unacceptable, you can add a check to prevent this from happening. The cost is that it needs to be done every time the user clicks on the button.

Fortunately, HTML5 provides the mouse events that we need for this application. In the **init** function, I include the following lines:

```
canvas1 = document.getElementById('canvas');

canvas1.onmousedown = function () { return false; };

canvas1.addEventListener('dblclick',makenewitem,false);

canvas1.addEventListener('mousedown',startdragging,false);
```

The first statement sets the canvas1 variable to reference the canvas element. The second statement is necessary to turn off the default action for the cursor. I also included a style directive for the canvas, which made the positioning absolute and then positioned the canvas 80 pixels from the top. This is ample space for the directions and the buttons.

```
canvas {position:absolute; top:80px;  
cursor:crosshair;  
}
```

The third and fourth statements set up event handling for double click and **mouse button down** events. We should appreciate the fact that we as programmers do not have to write code to distinguish mouse down, click and double click.

The **makenewitem** and the **startdragging** functions start off the same. The code first determines the mouse cursor coordinates and then loops through the **stuff** array to determine which, if any, object was clicked on. You probably have seen the mouse cursor coordinate code before, in the Essential Guide to HTML5, for example. The looping through the array is done in reverse order. Calls are made to the **overcheck** method, defined appropriately for the different types of objects. If there is a hit, then the **makenewitem** function calls the **clone** function to make a copy of that item. The code modifies the x and y slightly so the new item is not directly on top of the original. The new item is added to the array and there is a break to leave the **for** loop.

```
function makenewitem(ev) {  
  
var mx;  
  
var my;
```

```
if ( ev.layerX || ev.layerX == 0) {  
  
    mx= ev.layerX;  
  
    my = ev.layerY;  
  
} else if (ev.offsetX || ev.offsetX == 0) {  
  
    mx = ev.offsetX;  
  
    my = ev.offsetY; }  
  
var endpt = stuff.length-1;  
  
var item;  
  
for (var i=endpt;i>=0;i--) { //reverse order  
  
    if (stuff[i].overcheck(mx,my)) {  
  
        item = clone(stuff[i]);  
  
        item.x +=20;  
  
        item.y += 20; stuff.push(item);  
  
        break;  
    }  
}  
  
drawstuff();  
}
```

As I indicated earlier, the **clone** function makes a copy of an element in the **stuff** array. You may ask, why not just write

```
item = stuff[i];
```

The answer is that this assignment does not create a new, distinct value. JavaScript merely sets the **item** variable to point to the same thing as the *i*th member of **stuff**. This is called ‘copy by reference’. We don’t want that. We want a brand new, separate thing that we can change. The way to copy is demonstrated in the **clone** function. A new object is created and then a for-loop is invoked. The **for(var info in obj)** says: for every attribute of **obj** , set an equivalently named attribute in **item** to the value of the attribute.

```
function clone(obj) {  
  
  var item = new Object();  
  
  for (var info in obj) {  
  
    item[info] = obj[info];  
  
  }  
  
  return item;  
  
}
```

So the effect of the two functions is to duplicate whatever element is under the mouse cursor. You or your end-user can then mouse down on the original

or the cloned object and move it around.

The **startdragged** function proceeds as indicated to determine what object was under the mouse. The code then determines what I (and others) call the offsets in x and y of the mouse coordinates versus the x, y position of the object. This is because we want the object to move around maintaining the same relationship between object and mouse. Some folks call this the flypaper effect. It is as if the mouse cursor came down on the object and stuck like flypaper. The **offsetx** and **offsety** are global variables. Note that the coding works for objects for which the x, y values refer to the upper left corner (pictures and rectangles), to the center (ovals) and to a specific internal point (hearts).

The coding then performs a series of operations that has the effect of moving this object to the end of the array. The first statement is a copy by reference operation to set the variable item. The next step saves the index for the last element of the stuff array to the global variable **thingInMotion**. This variable will be used by the function **moveit**. The **splice** statement removes the original element and the **push** statement adds it to the array at the end. The statement referencing cursor is the way to specify a cursor. The “pointer” refers to one of the built-in options. The last two statements in the function set up the event handling for moving the mouse and releasing the button on the mouse. This event handling will be removed in the **dropit** function.

```
function startdragging(ev) {  
  
var mx;  
  
var my;  
  
if ( ev.layerX || ev.layerX == 0) {  
  
mx= ev.layerX;  
  
my = ev.layerY;  
  
} else if (ev.offsetX || ev.offsetX == 0) {  
  
mx = ev.offsetX;  
  
my = ev.offsetY; }  
  
var endpt = stuff.length-1;  
  
for (var i=endpt;i>=0;i--) { //reverse order  
  
if (stuff[i].overcheck(mx,my)) {  
  
offsetx = mx-stuff[i].x;  
  
offsety = my-stuff[i].y; var item = stuff[i];  
  
var last = stuff.length-1;  
  
stuff.splice(i,1);  
  
stuff.push(item);  
  
thingInMotion = last;
```

```
    canvas1.style.cursor = "pointer"; // change to finger

    canvas1.addEventListener('mousemove',moveit,false);
    canvas1.addEventListener('mouseup',dropit,false);

    break;

}

}

}
```

The **moveit** function moves the object referenced by **thingInMotion** and uses the **offsetx** and **offsety** variables to move the object. The **drawstuff** function is invoked to show the modified canvas.

```
function moveit(ev) {

var mx;

var my;

if ( ev.layerX || ev.layerX == 0) {

mx= ev.layerX;

my = ev.layerY;

} else if (ev.offsetX || ev.offsetX == 0) {

mx = ev.offsetX;
```

```
my = ev.offsetY;  
}  
  
stuff[thingInMotion].x = mx-offsetx; //adjust for flypaper dragging  
stuff[thingInMotion].y = my-offsety;  
drawstuff();  
}
```

A mousemove event is triggered if the mouse moves a single pixel in any direction. If this seems too much, remember that the computer does it, not you or I. The user gets a smooth response to moving the mouse.

The dropit function is invoked at a mouseup event. The response is to remove, stop the listening for moving and releasing the mouse and then changing the cursor back to the crosshairs.

Function

```
dropit(ev) {  
  
canvas1.removeEventListener('mousemove',moveit,false);  
canvas1.removeEventListener('mouseup',dropit,false);  
canvas1.style.cursor = "crosshair"; //change back to crosshair  
}
```

To summarize, the user interface for this application involves two buttons and two types of mouse actions. The drag and drop operation is implemented using a set of functions.

Saving the canvas to an image

After creating a composition, the user may want to save it to an image file. The Firefox browser makes this easy. You can right-click on the canvas when using a PC or do the equivalent operation on a MAC and a pop-up menu will appear with the option to Save Image As... However, Chrome, Safari and Opera do not provide that facility. If you right-click, the options concern the HTML document. There is, however, an alternative provided in HTML5.

A canvas element has a method called `toDataURL` that will produce an image from the canvas. The method provides a choice of image file types from among png, jpg, or bmp. What I choose to do with the result of this operation is write code to open a new window with the image as the content. The user then can save this image as a file either by the save file option or the right-click for the image. However, there is one more consideration. Chrome and Firefox require that this code run from a server, not on the client computer. The client computer is the one running the browser program. The server computer would be the website to which you will upload your finished work. You may or may not have one. Opera and Safari allow the code to run from the client computer. This has an impact on testing, since, generally speaking, we test programs locally and then upload to a server. Because of this situation, this is an appropriate place to use the try/catch facility of JavaScript for catching errors (so to speak) for the programmer to take action. Here is the code for the

saveasimage function. The variable canvas1 has been set to the canvas element in the init function invoked when the document is loaded.

```
function saveasimage() {  
    try {  
        window.open(canvas1.toDataURL("image/png"));}  
    catch(err) {  
        alert("You need to change browsers OR upload the file to a  
server.");  
    }  
}
```

Building the application and making it your own

You can make this application your own by identifying your own image files, specifying what rectangles, ovals and hearts you want to include in the collection of objects to be manipulated and, after you have something working, adding new object types. The application has many functions but they each are small and many share attributes with others. An informal summary / outline of the application is

1. *init for initialization, including setting up event handling for double click, mouse down, mouse move and mouse up*
2. *object definition methods: constructor functions, draw functions and overcheck functions*
3. *event handling functions: mouse events and button onClick*

More formally, Table 2-1 lists all the functions and indicates how they are invoked and what functions they invoke. Notice that several functions are invoked as a result of the function being specified as a method of an object type.

Table 2-1. Functions in the HTML5 Family Card project

Function	Invoked / Called By	Calls
init	invoked by action of the onLoad attribute in the <body> tag	
saveasimage	invoked by action of the onClick attribute in a button tag	
removeobj	invoked by action of the onClick attribute in a button tag	Picture, Rect, Oval, Heart, drawstuff
Picture	invoked in init function	
Rect	invoked in init function	drawstuff
Oval	invoked in init function	
Heart	invoked in init function	
drawheart	invoked in drawstuff	

Continued

Function	Invoked / Called By	Calls
drawrect	invoked in drawstuff	
drawoval	invoked in drawstuff	
drawpic	invoked in drawstuff	
overheart	invoked in startdragging and makenewitem	
overrect	invoked in startdragging and makenewitem	distsq, outside
overoval	invoked in startdragging and makenewitem	distsq
distsq	invoked by overheart and overoval	
drawstuff	invoked by makenewitem, moveit, removeobj, init	
moveit	invoked by action set by addEventListener for mousemove set in startdragging	draw method of each item in the stuff array
dropit	invoked by action set by addEventListener for mouseup set in startdragging	

outside	invoked by overheart	
makenewitem	invoked by action set by addEventListener for dblclick set in init	
clone	invoked by makenewitem	clone
startdragging	invoked by action set by addEventListener for mousedown set in init	

Table 2-2 shows the code for the basic application, with comments for each line.

Table 2-2. Complete Code for the HTML5 Logo project

Code Line	Description
<!DOCTYPE html >	standard heading for HTML5 documents
<html>	html tag
<head>	head tag
<title>Build family picture</title>	complete title
<meta charset="UTF-8">	meta tag
<style>	start of style

<code>canvas {position:absolute; top:80px;</code>	directive for canvas, setting its position as absolute and its location 80 pixels from the top of the document.
<code>cursor:crosshair;</code>	specifying the cursor icon for when the mouse is over the canvas
<code>}</code>	close directive
<code></style></code>	close style
<code><script language="Javascript"></code>	script tag
<code>var ctx;</code>	variable to hold the canvas context
<code>var canvas1;</code>	variable to hold the canvas element
<code>var stuff = [];</code>	array for all the objects on the canvas
<code>var thingInMotion;</code>	reference to object being dragged
<code>var offsetx;</code>	horizontal offset for object being dragged
<code>var offsety;</code>	vertical offset for object being dragged
<code>function init() {</code>	function header for init

Code Line	Description

<code>canvas1 = document.getElementById('canvas');</code>	sets variable to reference the canvas element
<code>canvas1.onmousedown = function () { return false; } ;</code>	prevents change of cursor to default
<code>canvas1.addEventListener('dblclick',makenewitem,false);</code>	sets up the event handling for double clicks on the canvas
<code>canvas1.addEventListener('mousedown',startdragging,false);</code>	sets up the event handling for mouse down on the canvas
<code>ctx = canvas1.getContext("2d");</code>	sets ctx to reference the context of the canvas. Used for all drawing.
<code>var r1 = new Rect(2,10,50,50,"red");</code>	constructs a rectangle
<code>var s1 = new Rect(60,10, 50,50,"blue");</code>	constructs a rectangle
<code>var oval1 = new Oval(200,30,20,2.0,1.0, "teal");</code>	constructs an oval
<code>var cir1 = new Oval(300,30,20,1.0,1.0,"purple");</code>	constructs an oval (which will be a circle)
<code>var dad = new Image();</code>	creates an Image element
<code>dad.src = "daniel1.jpg";</code>	sets the src to the indicated file

<code>var mom = new Image();</code>	creates an Image element
<code>mom.src = "allison1.jpg";</code>	sets the src to the indicated file
<code>var son1= new Image();</code>	creates an Image element
<code>son1.src = "liam2.jpg";</code>	sets the src to the indicated file
<code>var son2 = new Image();</code>	creates an Image element
<code>son2.src = "grant1.jpg";</code>	sets the src to the indicated file
<code>var pic1 = new Picture(10,100,100,100,dad);</code>	constructs a Picture object

Code Line	Description
<code>var pic2 = new Picture(120,100,100,100,mom);</code>	constructs a Picture object
<code>var pic3 = new Picture(230,100,100,100,son1);</code>	constructs a Picture object
<code>var pic4 = new Picture(340,100,100,100,son2);</code>	constructs a Picture object

<code>var heart1 = new Heart(400,30,60,20,"pink");</code>	constructs a Heart object
<code>stuff.push(pic1);</code>	adds (pushes) pic1 to stuff array
<code>stuff.push(pic2);</code>	adds pic2
<code>stuff.push(pic3);</code>	adds pic3
<code>stuff.push(pic4);</code>	adds pic4
<code>stuff.push(r1);</code>	adds r1
<code>stuff.push(s1);</code>	adds s1
<code>stuff.push(oval1);</code>	adds oval1
<code>stuff.push(cir1);</code>	adds cir1
<code>stuff.push(heart1);</code>	adds heart1
<code>drawstuff();</code>	draws all the objects on the canvas
<code>}</code>	end init function
<code>function distsq (x1,y1,x2,y2) {</code>	function header for distsq. Takes 2 points (2 x 2 values) as parameters
<code>var xd = x1 - x2;</code>	set difference in x
<code>var yd = y1 - y2;</code>	set difference in y
<code>return ((xd*xd) + (yd*yd));</code>	returns sum of squares. This is the square of the distance between the two points.

```
}
```

```
end distsq function
```

Code Line	Description
function Picture(x,y,w,h,imagename) {	function header for Picture constructor, positioned at x, y, with width w and height h, and the imagename Image object.
this.x = x;	set attribute
this.y = y;	set attribute
this.w = w;	set attribute
this.h = h;	set attribute
this.imagename = imagename;	set attribute
this.draw = drawpic;	set drawpic function to be the draw method
this.overcheck = overrect;	set overrect function to be the overcheck method
}	close function
function Heart(x,y,h,drx,color) {	function header for Heart constructor, located with the cleavage at x, y, distance from x, y to lower tip h, radius

	drx and color.
this.x = x;	set attribute
this.y = y;	set attribute
this.h = h;	set attribute
this.drx = drx;	set attribute
this.radsq = drx*drx;	set attribute to avoid doing this operation repeated times later
this.color = color;	set attribute
this.draw = drawheart;	set drawheart function to be the draw method

Code Line	Description
this.overcheck = overheart;	set overheart function to be the overcheck method
this.ang = .25*Math.PI;	set attribute to be this constant value. May make more general at a later time
}	close function

function drawheart() {	function header for drawheart
var leftctrx = this.x-this.drx;	calculate and set variable to be x coordinate of center of left curve
var rightctrx = this.x+this.drx;	calculate and set variable to be x coordinate of center of right curve
var cx = rightctrx+this.drx*Math.cos(this.ang);	calculate and set variable to be x coordinate of point where curve on the right changes to straight line
var cy = this.y + this.drx*Math.sin(this.ang);	calculate and set variable to be y coordinate of point where curve on the right changes to straight line
ctx.fillStyle = this.color;	set fillStyle
ctx.beginPath();	begin path

<code>ctx.moveTo(this.x,this.y);</code>	move to cleft of heart
<code>ctx.arc(leftctrx,this.y,this.drx,0,Math.PI-this.ang,true);</code>	draw left curve
<code>ctx.lineTo(this.x,this.y+this.h);</code>	move to bottom point
<code>ctx.lineTo(cx,cy);</code>	move to point where straight line meets curve
<code>ctx.arc(rightctrx,this.y,this.drx,this.ang,Math.PI,true);</code>	draw right curve
<code>ctx.closePath();</code>	close path

Code Line	Description
<code>ctx.fill();</code>	fill in path
<code>}</code>	close function
<code>function overheart(mx,my) {</code>	header for overheart function

<code>var leftctrx = this.x-this.drx;</code>	set variable to be x coordinate of center of left curve
<code>var rightctrx = this.x+this.drx;</code>	set variable to be x coordinate of center of right curve
<code>var qx = this.x-2*this.drx;</code>	calculate and set variable to be x coordinate of left of bounding rectangle
<code>var qy = this.y-this.drx;</code>	calculate and set variable to be y coordinate of top of bounding rectangle
<code>var qwidth = 4*this.drx;</code>	calculate and set variable to be width of bounding rectangle
<code>var qheight = this.drx+this.h;</code>	calculate and set variable to be height of bounding rectangle
<code>if (outside(qx,qy,qwidth,qheight,mx,my)) {</code>	quick test if it is in bounding rectangle
<code> return false;}</code>	
<code>if (distsq(mx,my,leftctrx,this.y)<this.radsq) return true;</code>	check if inside left curve
<code>if (distsq(mx,my,rightctrx,this.y)<this.radsq) return true;</code>	or right curve
<code>if (my<=this.y) return false;</code>	return false if above y on screen (and not previously determined to be within curves)

<code>var x2 = this.x</code>	start calculations to compare my to slopes. Set x2 and
<code>var y2 = this.y + this.h;</code>	set y2 to have x2,y2 point on each sloping line

Code Line	Description
<code>var m= (this.h)/(2*this.drx);</code>	calculate slope of left line
<code>if (mx<=this.x) {</code>	If mx is on the left side...
<code> if (my < (m*(mx-x2)+y2)) {return true;}</code>	compare my to the y value corresponding to mx. If my is above (on the screen), then return true
<code> else { return false;}</code>	otherwise return false
<code>}</code>	close if if (mx<=this.x) clause
<code>else {</code>	else
<code> m= -m;</code>	change sign of slope to be slope of the right line
<code> if (my < (m*(mx-x2)+y2)) { return true}</code>	Compare my to the value corresponding to mx on the right line and if less than (further up on the

	screen) return true
else return false;	else return false
}	close clause
}	close function
function outside(x,y,w,h,mx,my) {	function header outside
return ((mx < x) (mx > (x+w)) (my < y) (my > (y+h)));	returns true if any of factors is true, indicating the mx, my point is outside the rectangle
}	close function
function drawpic() {	function header drawpic
ctx.drawImage(this.imagename,this.x,this.y,this.w,this.h);	draw indicated image
}	close function

Code Line	Description
function Oval(x,y,r,hor,ver,c) {	function header for Oval constructor, position x, y, horizontal scaling hor, vertical scaling ver, color c.
this.x = x;	set attribute

this.y = y;	set attribute
this.r = r;	set attribute
this.radsq = r*r;	store as attribute to avoid repeated calculations later
this.hor = hor;	set attribute
this.ver = ver;	set attribute
this.draw = drawoval;	set drawoval as the draw method
this.color = c;	set attribute
this.overcheck = overoval;	set overoval as the overcheck method
}	close function
function drawoval() {	function header for drawoval
ctx.save() ;	save current coordinate state
ctx.translate(this.x,this.y);	move to center
ctx.scale(this.hor,this.ver);	scale as indicated by attributes
ctx.fillStyle = this.color ;	set color

<code>ctx.beginPath();</code>	start path
<code>ctx.arc(0,0,this.r,0,2*Math.PI,true);</code>	draw arc (complete circle)
<code>ctx.closePath();</code>	close path
<code>ctx.fill();</code>	fill in

Code Line	Description
<code>ctx.restore();</code>	restore original coordinate state
<code>}</code>	close function
<code>function Rect(x,y,w,h,c) {</code>	function header Rect constructor: position x,y, width w and height h, color c.
<code> this.x = x;</code>	set attribute
<code> this.y = y;</code>	set attribute
<code> this.w = w;</code>	set attribute
<code> this.h = h;</code>	set attribute
<code> this.draw = drawrect;</code>	set drawrect as the draw method

<code>this.color = c;</code>	set attribute
<code>this.overcheck = overrect;</code>	set overrect as the overcheck method
<code>}</code>	close function
<code>function overoval(mx,my) {</code>	function header for overoval
<code> var x1 = 0;</code>	set variable to be used in call to distsq. This represents x coordinate of point at center of oval
<code> var y1 = 0;</code>	set variable to be used in call to distsq. This represents y coordinate of point at center of oval
<code> var x2 = (mx-this.x)/this.hor;</code>	calculate the x2 using input and scaling factor
<code> var y2 = (my-this.y)/this.ver;</code>	calculate the y2 using input and scaling factor
<code> if (distsq(x1,y1,x2,y2)<=(this.radsq)){</code>	if distance squares is less than stored radius squared....
<code> return true</code>	return true

Code Line	Description

}	end clause
else {return false}	else return false
}	close function
function overrect (mx,my) {	function header for overrect
if ((mx>=this.x)&&(mx<=(this.x+this.w))&&(my>=this.y) &&(my<=(this.y+this.h))	If mx, my within bounds (the 4 sides)
{return true;}	return true
else {return false;}	else return false
}	close function
	function header for makenewitem. Has

function makenewitem(ev) {	as a parameter an event ev set by JavaScript
var mx;	variable will hold x coordinate of mouse
var my;	variable will hold y coordinate of mouse
if (ev.layerX ev.layerX == 0) {	does this browser use layer...
 mx= ev.layerX;	... set mx
 my = ev.layerY;	... my
} else if (ev.offsetX ev.offsetX == 0) {	does browser use offset...
 mx = ev.offsetX;	...set mx
 my = ev.offsetY;	... set my

}	end clause
var endpt = stuff.length-1;	store index of last item in stuff array

Code Line	Description
var item;	will hold the new item
for (var i=endpt;i>=0;i--) {	start search from the end
if (stuff[i].overcheck(mx,my)) {	is the mouse over this member of stuff
item = clone(stuff[i]);	clone (make copy of)
item.x += 20;	move over slightly horizontally
item.y += 20;	and vertically
stuff.push(item);	add newly created item to stuff array
break;	leave for loop
}	end if clause

}	end for loop
drawstuff();	draw everything
}	close function
function clone(obj) {	function header for clone
var item = new Object();	create an Object
for (var info in obj) {	loop over all attributes of the obj passed as parameter
item[info] = obj[info];	set an attribute by that name to the attribute value
}	close for loop
return item;	return the newly created object
}	close function
function startdragging(ev) {	function header for startdragging. Has as a parameter an event ev set by JavaScript

Code Line	Description

var mx;	variable will hold x coordinate of mouse
var my;	variable will hold y coordinate of mouse
if (ev.layerX ev.layerX == 0) { // Firefox, ???	does this browser use layer...
 mx= ev.layerX;	... set mx
 my = ev.layerY;	... my
} else if (ev.offsetX ev.offsetX == 0) {	does browser use offset...
 mx = ev.offsetX;	...set mx
 my = ev.offsetY;	... set my
}	end clause
var endpt = stuff.length-1;	store index of last item in stuff array
for (var i=endpt;i>=0;i--) {	start search from the end
 if (stuff[i].overcheck(mx,my)) {	is the mouse over this member of stuff
 offsetx = mx-stuff[i].x;	calculate how far the mx was from the x of this object
 offsety = my-stuff[i].y;	calculate how far the my was from the y of this object
 var item = stuff[i];	will now move this item to the end of the array. Set item

thingInMotion = stuff.length-1;	set global variable to be used in the dragging
stuff.splice(i,1);	remove this item from its original location
stuff.push(item);	add item to the end
canvas1.style.cursor = "pointer";	change cursor to finger when dragging

Code Line	Description
canvas1.addEventListener('mousemove', moveit, false);	set up event handling for moving the mouse
canvas1.addEventListener('mouseup', dropit, false);	set up event handling for releasing mouse button
break;	leave the for loop
}	close if clause
}	close for loop

}	close function
function dropit(ev) {	function header for dropit. Has as a parameter an event ev set by JavaScript
canvas1.removeEventListener('mousemove',moveit,false);	Remove (stop) event handling for moving the mouse
canvas1.removeEventListener('mouseup',dropit,false);	Remove (stop) event handling for releasing the mouse button
canvas1.style.cursor = "crosshair";	change cursor back to crosshair
}	close function
function moveit(ev) {	function header for moveit. Has as a parameter an event ev set by JavaScript
var mx;	variable will hold x coordinate of mouse

<code>var my;</code>	variable will hold y coordinate of mouse
<code>if (ev.layerX ev.layerX == 0) {</code>	does this browser use layer...
<code> mx= ev.layerX;</code>	... set mx
<code> my = ev.layerY;</code>	... my
<code>} else if (ev.offsetX ev.offsetX == 0) {</code>	does browser use offset...

Code Line	Description
<code>mx = ev.offsetX;</code>	...set mx
<code>my = ev.offsetY;</code>	... set my
<code>}</code>	end clause
<code>stuff[thingInMotion].x = mx-offsetx;</code>	set x for the thingInMotion, adjust for flypaper dragging
<code>stuff[thingInMotion].y = my-offsety;</code>	set y for the thingInMotion, adjust for flypaper dragging

drawstuff();	draw everything
}	close function
function drawstuff() {	function header for drawstuff
 ctx.clearRect(0,0,600,400);	clear (erase) canvas
 ctx.strokeStyle = "black";	set color for frame
 ctx.lineWidth = 2;	set lineWidth
 ctx.strokeRect(0,0,600,400);	draw frame
 for (var i=0;i<stuff.length;i++) {	iterate through the stuff array
 stuff[i].draw();	invoke the draw method for each member of the array
 }	close for
}	close function
function drawrect() {	function header drawrect
 ctx.fillStyle = this.color;	set the color
 ctx.fillRect(this.x, this.y, this.w, this.h);	draw a filled rectangle
}	close function

Code Line	Description

function saveasimage() {	function header for saveasimage
try {	start try clause
window.open(canvas1.toDataURL("image/png"));}	create the image data and use it as contents of new window
catch(err) {	if that didn't work, that is, threw an error
alert("You need to change browsers OR upload the file to a server.");	display alert message
}	close catch clause
}	close function
function removeobj() {	function header for removeobj
stuff.pop();	remove the last member of the stuff array
drawstuff();	draw everything
}	close function
</script>	close script element
</head>	close head element
<body onLoad="init();">	body tag, with onLoad set
Mouse down, move and mouse up to move objects.	
Double click for	

new object.
	Text giving directions
<canvas id="canvas" width="600" height=400">	canvas tag
Your browser doesn't recognize the canvas element	message for older browsers
</canvas>	ending canvas tag

Code Line	Description
<button onClick="saveasimage();">Open window with image (which you can save into image file)</button> 	button for saving image
<button onClick="removeobj();">Remove last object moved</button>	button for removing object
</body>	close body tag
</html>	close html tag

It is obvious how to make this application your own using only the techniques demonstrated in my example: gather photos of your own family or acquire other photographs and use the Rect, Oval, and Heart to create your own set of shapes.

You can define your own objects, using the coding here as a model. For example,

the *Essential Guide to HTML5* book included coding for displaying polygons. You can make the overcheck function for the polygon treat the polygon as a circle, perhaps a circle with smaller radius, and your customers will not object.

The next step could be to build an application that allows the end-user to specify the addresses of image files. You would need to set up a form for doing this. Another enhancement is to allow the end-user to enter text, perhaps a greeting, and position it on the canvas. You would create a new object type and write the **draw** and **overcheck** methods. The **overcheck** method could be **overrect**, that is, the program accepts as being on the text anything in the bounding rectangle.

Testing and uploading the application

You need to gather all the image files you want to include in your application. The testing procedure depends on what browser you are using. Actually, it is a good practice to test with several browsers. If you are using Firefox or Chrome, you need to upload the application: the html file and all image files, to a server to test the feature for creating an image. However, the other aspects of the application can be tested on your own [client] computer.

Summary

In this chapter, you learned how to build an application involving creating and positioning specific shapes, namely rectangles, ovals and hearts, along with pictures such as photographs on the canvas. The programming techniques and HTML5 features included

- programming-defined objects
- mouse events on canvas
- try and catch for trapping errors
- algebra and geometry for several functions.

The next chapter describes creating an application showing a video clip bouncing around like a ball in a box.

CHAPTER 3

Bouncing Video: Animating and Masking HTML5

Video:

In this chapter, you will learn how to do the following:

- Produce a moving video clip by drawing the current frame of the video at different locations on a canvas
- Produce a moving video clip by repositioning the video element in the document
- Mask the video so it looks like a circle for both situations
- Build an application that will adapt to different window sizes

Introduction

The project for this chapter is a display of a video clip in the shape of a ball bouncing in a box. An important new feature in HTML5 is the native support of video (and audio). The book *The Definitive Guide to HTML5 Video*, by Silvia Pfeiffer (Apress, 2010), is an excellent reference. The challenge in this project is making the video clip move on the screen. I will describe two different ways to implement the application. The screenshots do not reveal the differences.

Figure 3-1 shows what the application looks like in the full-window view in Opera. The video is a standard rectangular video clip. It appears ball-like because of my coding. You can skip ahead to Figure 3-8 to get an idea of the mask created to ride along with the video. All the figures are static screen captures of animations. You need to take my word for it that the video does move and bounce within the box.

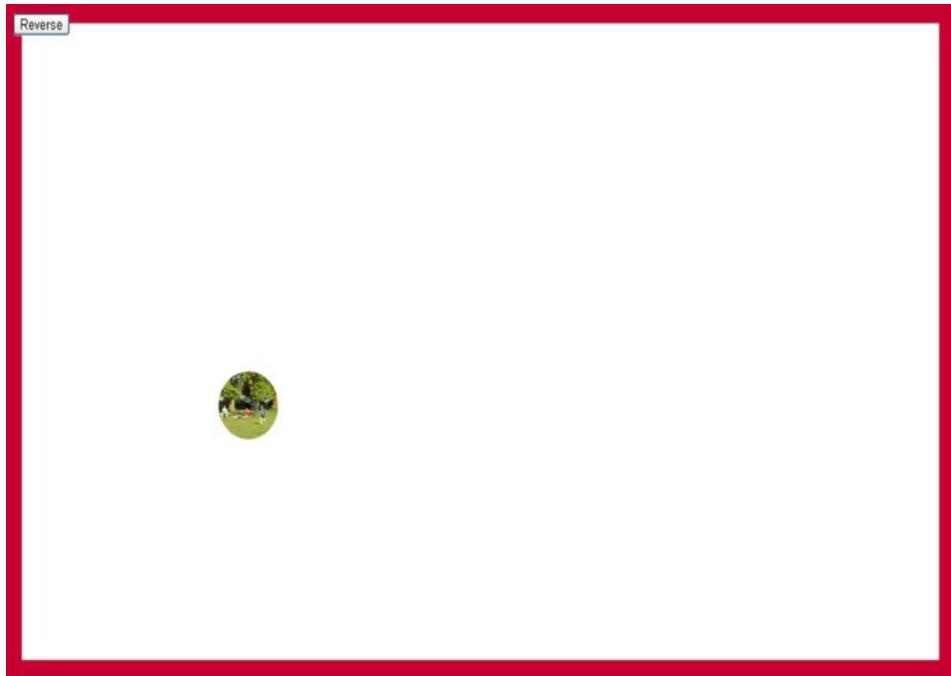


Figure 3-1. Screen capture, full window

In all cases, when the virtual ball hits a wall, it appears to bounce off the wall. If, for example, the virtual ball is moving down the screen and to the right, when it hits the right side of the box, it will head off to the left but still

moving down the screen. When the virtual ball then hits the bottom wall of the box, it will bounce to the left, heading up the screen. The trajectory is shown in Figure 3-2. To produce this image, I changed the virtual ball to be a simple circle and did not write code to erase the canvas at each interval of time. You can think of it as stop-motion photography. Changing the virtual ball was necessary because of its complexity: an image from a video clip and an all-white mask. I include the code for the trajectory program in the “Building the Application and Making It Your Own” section.

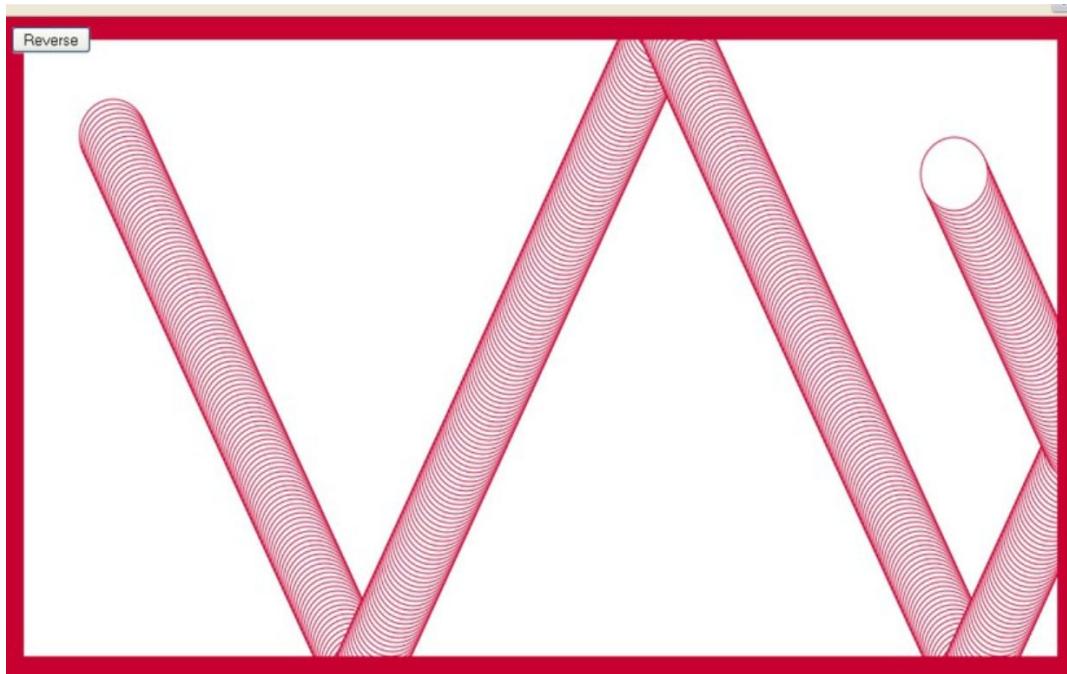


Figure 3-2. Trajectory of virtual ball

If I resize the browser window to be a little bit smaller and reload the application, the code will resize the canvas to produce what is shown in Figure 3-3: a smaller box.

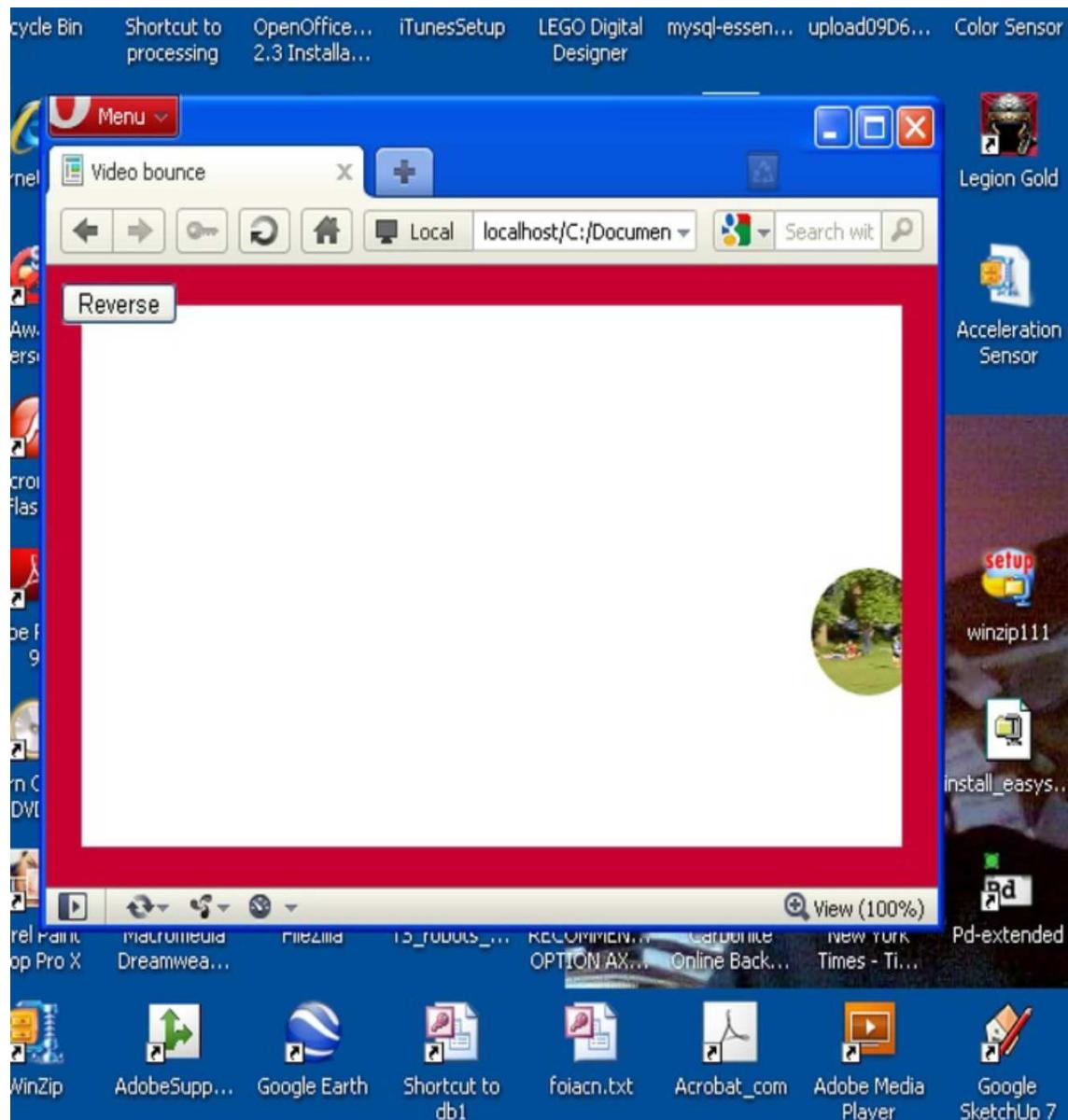


Figure 3-3. Application in smaller window

If the window is made very small, this forces a change in the size of the video clip itself, as well as the canvas and the box, as shown in Figure 3-4.



Figure 3-4. Window resized to very small

The application adapts the box size, and possibly the virtual video ball size, to the window dimensions at the time that the HTML document is first loaded. If the window is resized by the viewer later, during the running of the application, the canvas and video clip are not resized. In this case, you would see something like Figure 3-5, a small box in a big window.

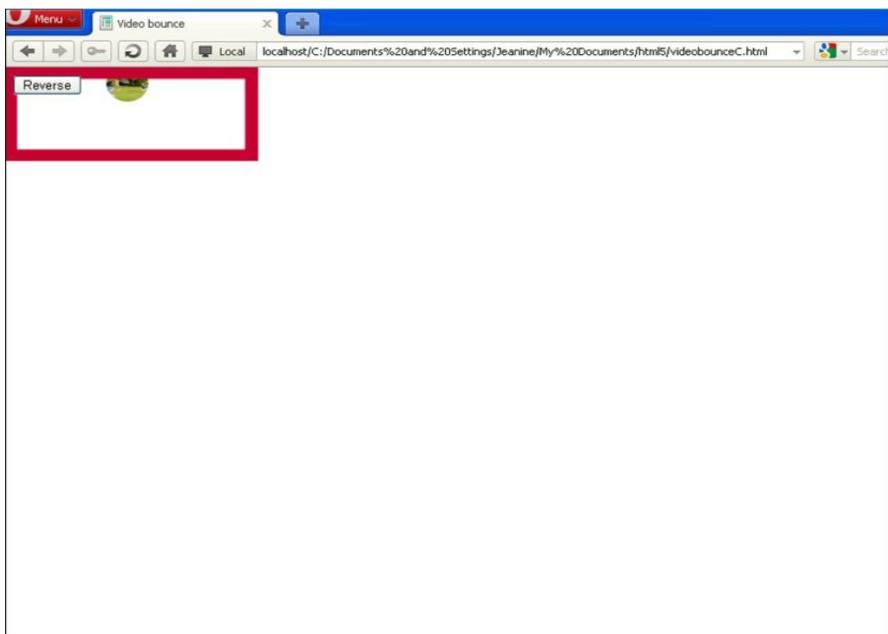


Figure 3-5. Window resized during running to be larger

Similarly, if you start the application using a full-size window, or any large window, and resize it to something smaller during the running of the program, you would see something like Figure 3-6, where the scroll bars are displayed by the browser to indicate that the content of the document is wider and longer than the window. If you, the viewer, choose not to use the scroll bars, then the video clip will disappear out of sight periodically for a short period of

time before reappearing

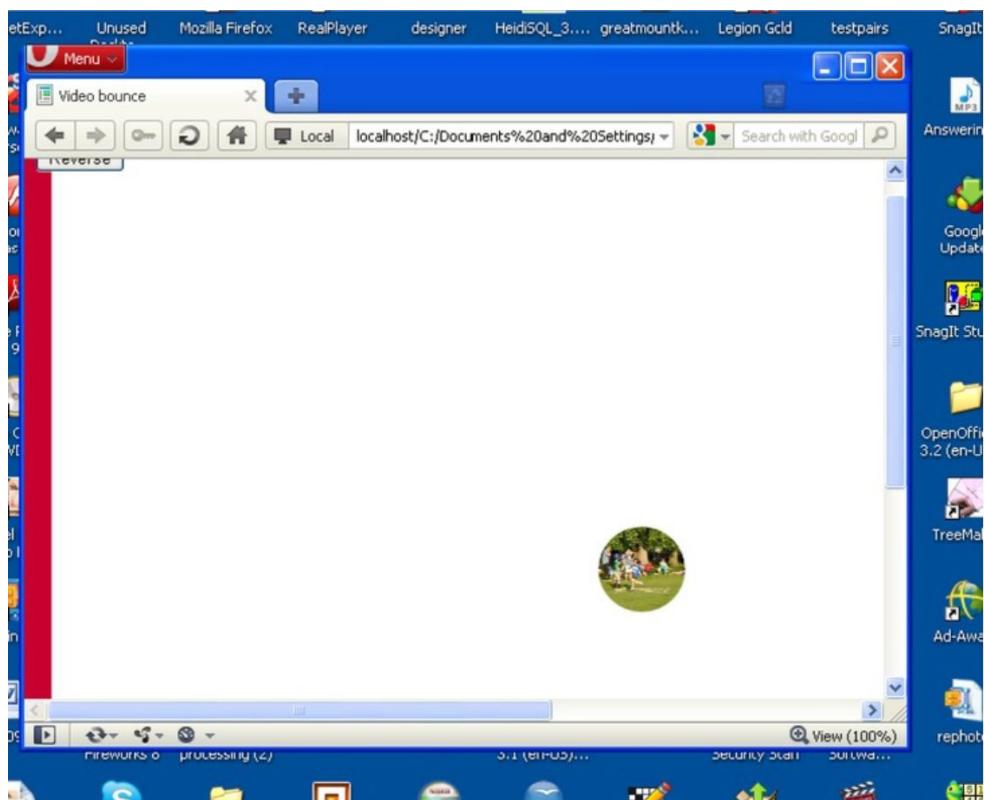


Figure 3-6. Large window resized

The two applications (I named them `videobounceC` for “video drawn on canvas” and `videobounceE` for “video element”) have been tested successfully in Firefox, Chrome, and Opera. The project demonstrates coding techniques using HTML5, JavaScript, and CSS for manipulating video and using video together with the canvas for special effects. The project also explains calculations that are helpful in customizing applications to the dimensions of the browser window.

Project History and Critical Requirements

I have always liked the application of simulating a ball bouncing in a box. Chapter 3 in *The Essential Guide to HTML5* features projects showing a ball produced by a path drawing and a ball produced by an image, each bouncing in a two-dimensional enclosure. I decided I wanted to make a video clip do the same thing. My explanation of the coding is complete in this chapter. However, if the first book is available to you (shameless plug), you may benefit from seeing what is the same and what is different among the various versions. In the ball and image applications, the canvas was set to fixed dimensions and was located with other material in the document. Because I did not want my video clip to be too small, I decided to use the whole window in this case. This objective produced the challenge of determining the dimensions of the document window. In the old ball and image applications, I wanted to demonstrate form validation, so the program provided form elements to change the vertical and horizontal speed. For the bouncing ball video clip, the application just provides one action for the user: a button to reverse direction. After studying this chapter, you should be able to add the other interface operations to the video application.

Since the ability to draw video on canvas is a feature of HTML5, this was the first approach I took for doing the project. However, my reading indicated that this is considered too much of a computation hog and is something to be

avoided, so I also developed another approach: moving a video element and moving an element on a canvas on top of it.

Putting off the implementation details, the critical requirements, in addition to determining the dimensions of the window, are to move—reposition—and animate a video clip simultaneously with a graphical element that acts as a mask. The effect of the mask is to make the video clip appear as a circle instead of the standard rectangular shape. The video clip is playing while it is moving. The application is to simulate a ball-like object bouncing within a box. Therefore, the application must display the walls of the box and perform calculations so that when the video clip appears to collide with any of the walls, the direction of motion changes in the appropriate way. A fancy way to describe the change is that the angle of reflection must equal the angle of incidence. In practical terms, what it means is that when the video clip virtually hits the bottom or top walls, it keeps going in the same direction horizontally (to the left if it was traveling to the left and to the right if it was traveling to the right), but switches direction vertically. When the video clip virtually hits either the left or the right wall, it keeps going in the same direction vertically (traveling up if it was traveling up and traveling down if it was traveling down), but switches direction horizontally. If you are interested in simulating real-life physics, you can slow down the motion at each virtual hit of a wall.

HTML5, CSS, and JavaScript Features

Any order of explanation means something is often discussed before the reason for doing it is clear. In this section, I will show how certain variables are set that will be shown in use later on. The general plan is to extract the window dimensions to set variables for the canvas and the video clip that will be referenced in the coding for drawing the video and the mask.

Definition of the Body and the Window Dimensions

The Document Object Model (DOM) provides information about the window in which the HTML document is displayed by the browser. In particular, the attributes **window.innerWidth** and **window.innerHeight** indicate the usable dimensions of the window. My code will use these values when it sets up the application.

Recall that the HTML5 video element can contain as child elements any number of source elements referencing different video files. At this time, this is necessary because the browsers that recognize the video element do not accept the same video formats (codecs). The situation may change in the future.

If you know the browser used by *all* your potential customers, you can determine a single video format. If that is not the case, you need to make three versions of the same video clip. The Open Source Miro Video Converter, downloadable from www.mirovideoconverter.com/, is a good product to convert a video clip into other formats.

With that reminder, I can present the body element for this application. It contains a video element, a button, and a canvas element:

```
<body onLoad="init();">  
  
<video id="vid" loop="loop" preload="auto">  
  
<source src="joshuahomerun.mp4" type='video/mp4';
```

```
codecs="avc1.42E01E, mp4a.40.2">'>  
  
<source      src="joshuahomerun.webmvp8.webm"      type='video/webm;  
codec="vp8, vorbis"'>  
  
<source      src="joshuahomerun.theora.ogv"      type='video/ogg;  
codecs="theora, vorbis"'>  
  
Your browser does not accept the video tag. </video>  
  
<button id="revbtn" onClick="reverse();">Reverse </button><br/>  
  
<canvas id="canvas" >  
  
This browser doesn't support the HTML5 canvas element.  
  
</canvas>  
  
</body>
```

Style directives will change the location of the three elements: video, canvas, and button.

In the **init** function invoked when the document is loaded, the following statements set the dimensions of the canvas to match the dimensions of the window:

```
canvas1 = document.getElementById('canvas');
```

```
ctx = canvas1.getContext('2d');
```

```
canvas1.width = window.innerWidth;  
cwidth = canvas1.width;  
canvas1.height = window.innerHeight;  
cheight = canvas1.height;
```

These statements also set variables that will be used later. So the task of adapting the canvas to the window is accomplished.

Now the next task takes more thought. How much do I want to adapt the video to the window dimensions? I decided that I would reduce the video width and height to one-third of the original width and height in all situations. However, I would reduce it further if the window were very small. The **Math.min** method returns the smallest of its operands, so the statements

```
v = document.getElementById("vid");  
v.width = Math.min(v.videoWidth/3,.5*cwidth);  
v.height = Math.min(v.videoHeight/3,.5*cheight);
```

start by setting the variable **v** to point to the video element, which you can see I have coded in the body to have the **id "vid"**. It then sets the **width** of the video to either one-third of the original, intrinsic width of the video clip or to half the width of the canvas, whichever is less. The next statement does the same for the height of the video. This approach does not make the video clip

proportional to the canvas. When you are working on this or another application, you will need to decide the approach you want to take.

Certain other variables are set in the **init** function and used for the drawing of the box and for the mask. The code is

```
videow = v.width;
```

```
videoh = v.height;
```

```
ballrad = Math.min(50,.5*videow,.5*videoh);
```

```
maskrad = .4*Math.min(videow,videoh); ctx.lineWidth = ballrad;
```

The **ballrad** variable is inherited from the previous applications. It is the radius of the ball, assuming the video height and width are equal and less than 50 pixels. Its use in the videobounce applications is to set the width of the line used to draw the box. The radius of the hole in the mask is set to .4 of the minimum of the width and height of the video. You certainly can experiment with these values and also experiment with the shape of the mask.

Animation

Animation is the trick by which still images are presented in succession fast enough so that our eye and brain interprets what we see as motion. The exact mechanics of how things are drawn will be explained in the next two sections. Keep in mind that there are two animations going on: the presentation of the video and the location of the video in the box. In this section, I talk about the location of the video in the box.

The way to get animation in HTML and JavaScript is to use the **setInterval** function. This function is called with two parameters. The first is the name of a function that we want to call and the second indicates the length of the time interval between each call to the function. The unit of time is milliseconds.

The following statement, which is in the **init** function, sets up the animation:

```
setInterval(drawscene,50);
```

drawscene refers to a function that will do all the work. The **50** stands for 50 milliseconds. This means that every 50 milliseconds (or 20 times per second), the **drawscene** function will be invoked. Presumably, **drawscene** will do what needs to be done to display something showing the video clip at a new location. You can experiment with interval duration.

If you want to enhance this application or build another one in which it

makes sense to stop the animation, you would declare a local variable for the **setInterval** call (let's call it **tid**) and use the statement

```
tid = setInterval(drawscene,50);
```

At the point when you need to stop the animation, or more formally, stop the interval-timing event, you code

```
clearInterval(tid);
```

If you have more than one timing event, you would assign the output for each of them to a new variable. Be careful not to call **setInterval** multiple times with the same function. Doing so has the effect of adding new timing events and invoking the function multiple times.

Much of the details of the **drawscene** function will be described in the next sections. However, two critical tasks are erasing the canvas and then determining the next position of the video clip. The statement for erasing the whole canvas is

```
ctx.clearRect(0,0,cwidth,cheight);
```

Notice that it makes use of the **cwidth** and **cheight** values calculated based on the window dimensions.

The simulation of bouncing is performed by a function called **moveandcheck**. The position of the virtual ball is defined by the variables **ballx** and **bally**.

The `(ballx,bally)` position is the upper-left corner of the video. The motion, also termed the *displacement*, is defined by the variables `ballvx` and `ballvy`. These two variables are termed the horizontal and vertical displacements, respectively.

The objective of the `moveandcheck` function is to reposition the virtual ball by setting `ballx` and `bally`, and when appropriate, change the signs of `ballvx` and `ballvy`. The way the code works is to try out new values (see `nballx` and `nbally` in the function) and then set `ballx` and `bally`. Changing the sign of the displacement values has the effect of making the balls bounce by changing the appropriate horizontal or vertical adjustment on the next interval.

The task now is to determine when to do the bounce. You need to accept that as far as the computer's concerned, there are no balls, bouncing or otherwise, and no walls. There are just calculations. Moreover, the calculations are done at discrete intervals of time. There is no continuous motion. The virtual ball jumps from position to position. The trajectory appears smooth because the jumps are small enough and our eye-brain interprets the pictures as continuous motion. Since the walls are drawn after the video (that will be explained later), the effect is that the virtual ball touches and goes slightly behind the wall before changing direction.

My approach is to set up trial or stand-in values for `ballx` and `bally` and do calculations based on these values. You can think of it logically as asking, If the video ball were moved, would it be beyond any of the walls? If so, readjust to just hit that wall and change the appropriate displacement value. The new displacement value is not used immediately, but will be part of the calculation made at the next iteration of time. If the trial value is not at or beyond the wall, keep the trial value as it is and keep the corresponding displacement value as it is. Then change `ballx` and `bally` to the possibly adjusted stand-in values.

The function definition for the videobounceC program is

```
function moveandcheck() {  
  
    var nballx = ballx + ballvx +.5*videow;  
  
    var nbally = bally + ballvy +.5*videoh;  
  
    if (nballx > cwidth) {  
  
        ballvx = -ballvx;  
  
        nballx = cwidth;  
  
    }  
  
    if (nballx < 0) {  
  
        nballx = 0;  
  
        ballvx = -ballvx; }  
  
    if (nbally > cheight) { nbally = cheight;  
  
        ballvy = -ballvy;  
  
    }  
  
    if (nbally < 0) { nbally = 0;  
  
        ballvy = -ballvy;  
  
    }  
}
```

```
ballx = nballx-.5*videow;  
bally = nbally-.5*videoh;  
}
```

The **moveandcheck** function is slightly different for moving a video element because of an issue involving scrolling, which I will discuss later. The basic concepts are the same. The **moveandcheck** function in videobounceE is

```
function moveandcheck() {  
  
var nballx = ballx + ballvx;  
  
var nbally = bally + ballvy;  
  
if ((nballx+videow) > cwidth) {  
  
ballvx = -ballvx;  
  
nballx = cwidth-videow;  
  
}  
  
if (nballx < 0) {  
  
nballx = 0;  
  
ballvx = -ballvx;  
  
}  
  
if ((nbally+videoh) > cheight) {
```

```
nbally = cheight-videooh;
```

```
ballyv = -ballyv;
```

```
}
```

```
if (nbally < 0) {
```

```
nbally = 0;
```

```
ballyv = -ballyv;
```

```
}
```

```
ballx = nballx;
```

```
bally = nbally;
```

```
}
```

Notice that the videobounceC version compares **ballx + ballvx + .5*videow** to **cwidth**, whereas videobounceE compares **ballx + ballvx + videow** to **cwidth**. This means the videobounceE program will force bouncing sooner—that is, turn around sooner—when compared with the right wall. The same holds true for the checking against the bottom wall. I did this to avoid a problem involving automatic scrolling. The video element is not restricted to the canvas, so if it moves out from under the canvas, it is part of the document and is displayed. Because the new display is bigger than the window, this causes scrolling. The scroll bars would appear, and though you would not see

anything, I did not like the effect. If you started with a smaller window and made it larger during the program execution, you could see something like what is shown in Figure 3-7.

Figure 3-7. Video element bouncing with less restrictive checking

To avoid this, you will see that I changed the checking for the video element application. The downside to doing this is that the video ball barely touches the right and bottom walls.

The reason why these effects do not happen in the video-drawn-on-canvas application, videobounceC, is that drawing on canvas with coordinates outside of the canvas has no visible effect. You can look back to Chapter 1, Figure 1-6, to see an example of drawing “outside the lines” producing nothing outside the canvas.

Note There may be other ways to avoid the scrolling problem. This would not prevent the unsightliness shown in Figure 3-7. It may be possible to prevent scrolling in the browser. It is possible to stop the user from scrolling, but automatic scrolling appears to be more of a challenge.

Video Drawn on Canvas and As a Movable Element

I now describe two different implementations: one with material from the video drawn on the canvas and the other with the video element moved around the document.

Video Drawn on Canvas

As I mentioned previously, HTML5 does provide the facility to draw video on the canvas as one would draw an image. This actually is a misnomer. Video clips are made up of sequences of still images called *frames*. Frame rates vary but typically are 15 to 32 frames per second, so you can understand that video files tend to be large. Video is stored using different types of encodings, each of which may make different technical trade-offs in terms of quality and storage size. We do not need to be concerned with these technicalities, but can think of the video as a sequence of frames. Playing a video involves presenting the frames in sequence. What happens in the **drawImage** command is that the current frame of the video clip is the image drawn on the canvas. If this operation is performed through a timed interval event, then the viewer will see a frame at each interval of time. There is no guarantee that the images shown are successive frames from the video clip, but if done fast enough, the frames drawn

will be close enough to the actual sequence that our eye and brain experience it as the live action of the video clip.

The command in pseudocode is

`ctx.drawImage(video element, x position, y position, width, height);`

This command, formally a method of the **ctx** canvas context, extracts the

image corresponding to the current frame of the video and draws it at the x and y values, with the indicated width and height. If the image does not have the specified width and height, the image is scaled. This will not occur for this situation.

The goal is to make the traveling video clip resemble a ball. For this application, this means we want to mask out all but a circle in the center of the rectangular video clip. I accomplish this by creating a traveling mask. The mask is a drawing in the canvas. Since I want to place the video element on the canvas element, and also position a shape created by drawing a path on top of the image drawn from the video clip, I use CSS directives to make both video and canvas be positioned using absolute positioning. I want the Reverse button to be on top of the canvas. These directives do the trick:

```
#vid {position:absolute; display:none; }

#canvas {position:absolute; z-index:10; top:0px; left:0px;} #revbtn
{position:absolute; z-index:20; }
```

A way to remember how the layering works is to think of the z-axis as coming out of the screen. Elements set at higher values are on top of elements set at lower values. The top and left properties of the canvas are each set to 0 pixels to position the upper-left corner of the canvas in the upper-left corner of the window.

Note When the z-index is referenced or modified in JavaScript, its name is zIndex.

Hopefully, you appreciate why the name `z-index` would not work: the hyphen (-) would be interpreted as a minus operator.

The video element is set in the style directive to have no display. This is because as an element by itself, it is not supposed to show anything. Instead, the content of the current frame is drawn to the canvas using the following statement:

```
ctx.drawImage(v,ballx, bally, videow,videoh);
```

The `ballx` and `bally` values are initialized in the `init` functions and incremented as described in the last section. The width and height of the video clip have been modified to be appropriate for the window size.

One way to understand this is to imagine that the video is being played somewhere offscreen and the browser has access to the information so it can extract the current frame to use in the `drawImage` method.

Movable Video Element

The videobounceE application moves the actual video element on the document. The video element is not drawn on the canvas, but is a distinct element in the HTML document. However, I need to make sure that the mask, which is drawn on the canvas, is always in the right place with respect to the video element. I need to code the style directives to make sure that the video is under the canvas, which in turn is under the Reverse button. A critical step is to set the positioning to be absolute for all three elements (video, canvas, and button) and position the canvas so that it is located with its upper-left corner in the upper-left corner of the window. This is critical for positioning the video element and mask, as we shall explore later. The style directives are

```
#vid {position:absolute; display:none;z-index: 1; }

#canvas {position:absolute; z index:10; top:0px; left:0px;}

#revbtn {position:absolute; z-index:20;}
```

Moving a video element around requires making the video visible and starting the playing of the video. It also requires positioning. The video element is positioned through references to **style.left** and **style.top**. Furthermore, the settings for the **left** and **top** attributes must be in the form of a character string representing a number followed by the string "**px**" , standing for pixels.

The following code

```
v.style.left = String(ballx)+"px";  
v.style.top = String(bally)+"px";  
v.play();  
v.style.visibility = "visible";  
v.style.display = "block";
```

is executed in the **init** function. Notice also that the initial position of the video is changed to the initial **ballx** and **bally** values. The numeric values need to be converted to strings, and then the "**px**" needs to be concatenated to the ends of the strings. This is because HTML/JavaScript assumes that style attributes are strings. I write the same code for setting the video element's **top** and **left** properties to the values corresponding to **ballx** and **bally** in the drawscene function. The statements that replace the ctx.drawImage statement are

```
v.style.left = String(ballx)+"px";  
v.style.top = String(bally)+"px";
```

All the code for both videobounceC and videobounceE will be listed with comments in the “Building the Application and Making It Your Own” section.

Looping Video

You may have noticed that the video tag has the attribute setting **loop="loop"**. This indicates that the video is to loop—that is, start over again—each time the playing of the clip reaches the end. At the time of writing, this does not work for Firefox, so I use the statement

```
v.addEventListener("ended",restart,false);
```

to set up an event to invoke the indicated function when the **ended** event occurs. The **false** parameter means that the event should not be bubbled to any other application. It's unlikely that another application is listening for this event, but it doesn't hurt to stop the bubbling action. I defined the function called **restart**.

```
function restart() {  
  v.currentTime=0;  
  v.play();  
}
```

Traveling Mask

The objective of the mask is to mask out—that is, cover up—all of the video except for a circle in the center. The style directives ensure that I can use the same variables—namely **ballx** and **bally**—to refer to the video and mask in both situations: video drawn and video element moved. So now the question is how to make a mask that is a rectangular donut with a round hole.

I accomplish this by writing code to draw two paths and filling them in with white. Since the shape of the mask can be difficult to visualize, I have created two figures to show you what it is. Figure 3-8 shows the outline of the two paths



Figure 3-8. Outline of paths for the mask



Figure 3-9 shows the outline and the paths filled
in

Figure 3-9. Paths for the mask after a fill and a stroke

Now, the actual path only has the fill, and the fill color is white. You need to imagine these two white shapes traveling along on top of the video. The effect of the mask is to cover up most of the video clip. The parts of the canvas that have no paint on them, so to speak, are transparent, and the video clip content shows through. Putting it another way, the canvas is on top of the video element, but it is equivalent to a sheet of glass. Each pixel that has nothing drawn in it is transparent.

The code drawing the masks is the same for both video drawn on canvas (`videobounceC`) and video element moving on the screen (`videobounceE`). The first path starts at the upper-left corner, and then goes over, down to the midway point, and finally back left. The path then is a semicircular arc. The last parameter indicating the sense of the arc is **true** for counterclockwise. The path continues with a line to the left edge and then back up to the start. The second path starts in the middle of the left edge, proceeds down to the lower-left corner, goes to the lower-right corner, moves up to the middle of the right side, and then moves to the left. The arc this time has **false** as the value of the parameter for direction, indicating the arc is clockwise. The path ends where it started.

```
ctx.beginPath();
```

```
ctx.moveTo(ballx,bally);
```

```
ctx.lineTo(ballx+videow,bally);

ctx.lineTo(ballx+videow,bally+.5*videoh);

ctx.lineTo(ballx+.5*videow+maskrad, bally+.5*videoh);

ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0,Math.PI,true);

ctx.lineTo(ballx,bally+.5*videoh);

ctx.lineTo(ballx,bally);

ctx.fill();

ctx.moveTo(ballx,bally+.5*videoh);

ctx.lineTo(ballx,bally+videoh);

ctx.lineTo(ballx+videow,bally+videoh);

ctx.lineTo(ballx+videow,bally+.5*videoh);

ctx.lineTo(ballx+.5*videow+maskrad,bally+.5*videoh);

ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0,Math.PI,false);

ctx.lineTo(ballx,bally+.5*videoh);

ctx.fill();
```

You can follow along the coding with my “English” description to see how it works.

By the way, my initial attempt was to draw a path consisting of a four-sided

shape representing the outer rectangle and then a circle in the middle. This worked for some browsers, but not others.

For the videobounceC application, the mask is on top of the video drawn on canvas because the two white filled-in paths are drawn after the **drawImage** statement draws a frame from the video. I achieve the same effect in the videobounceE application by specifying the z-index of the video element to be 0 and the z-index of the canvas to be 10. Remember that the z-axis is the axis that comes out of the screen. Higher values are closer to us and on top of lower values. The canvas with the z-index set to 10 is on top of the video element with the z-index set to 0. The next chapter, which demonstrates a spotlight moving on top of a map from Google Maps, will feature changing the z-index using JavaScript.

User Interface

The user interface for both versions of the videobounce project only includes one action for the user: the user can reverse the direction of travel. The button is defined by an element in the body:

```
<button id="revbtn" onClick="reverse();">Reverse </button><br/>
```

The effect of the **onClick** setting is to invoke the function named **reverse**. This function is defined to change the signs of the horizontal and vertical displacements:

```
function reverse() { ballvx = -ballvx; ballvy = -ballvy;  
}
```

There is one important consideration for any user interface. You need to make sure it is visible. This is accomplished by the following style directive:

```
#revbtn {position:absolute; z-index:20; }
```

The z-index places the button on top of the canvas, which in turn is on top of the video.

Having explained the individual HTML5, CSS, and JavaScript features that can be used to satisfy the critical requirements for bouncing video, I'll now show the code in the two videobounce applications along with the code used to show the trajectory in Figure 3-2.

Building the Application and Making It Your Own

The two applications for simulating the bouncing of a video clip ball in a two-dimensional box contain similar code, as does the program that produced the picture of the trajectory. A quick summary of the applications follows. The video applications are summarized by the following:

1. **init**: initialization, including adapting to fit the window and setting up the timed event for invoking **drawscene**
2. **drawscene**:
 - a. Erase the canvas.
 - b. Determine new location of video (virtual ball) using **moveandcheck**.
 - c. Either draw the image from video at a specified location on the canvas or reposition the video element to a specified position.
 - d. Draw paths on canvas to act as a mask to the video.
 - e. Draw the box.
3. **moveandcheck**: Check if the virtual ball will hit any wall. If so, change the appropriate displacement value.

The trajectory function also uses **init** and **moveandcheck**, but has a simpler **drawscene** function:

1. Determine the new location of the circle (virtual ball) using

moveandcheck.

2. Draw a path that consists of a circle and draw the circle using fill and then stroke.
3. Draw the box.

The function describing the invoked/called by and calling relationships (shown in Table 3-1) are the same for all the applications.

Table 3-1. Functions in the Bouncing Video Projects

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the <body> tag	
drawscene	Invoked by action of the setInterval command issued in init	
moveandcheck	Invoked in drawscene	moveandcheck

reverse	Invoked by action of onClick in the button	
restart	Invoked by action of addEventListener in init (not present in videobounceTrajectory)	

Table 3-2 shows the code for the videobounceC application, which draws the current frame of the video on the canvas at set intervals of time.

Table 3-2. Complete Code for the VideobounceC Application

Code Line	Description
<!DOCTYPE html>	Header
<html>	Opening html tag
<head>	Opening head tag
<title>Video bounce</title>	Complete title
<meta charset="UTF-8">	Meta element
<style>	Opening style
	Set up positioning of video; set

<code>#vid {position:absolute; display:none; }</code>	display to none; video element never appears
<code>#canvas {position:absolute; z-index:10; top:0px; left:0px;}</code>	Set positioning to absolute and position to be upper-left corner; set z-index so it is under the Reverse button
<code>#revbtn {position:absolute; z-index:20; }</code>	Set positioning to absolute and z-index so it is over the canvas
<code></style></code>	Close style
<code><script type="text/javascript"></code>	Opening script tag
<code>var ctx;</code>	Used to hold canvas context, used for all drawing
<code>var cwidth ;</code>	Used to hold canvas width
<code>var cheight ;</code>	Used to hold canvas height
<code>var ballrad = 50;</code>	Set ball radius
<code>var ballx = 50;</code>	Initial horizontal coordinate for ball
<code>var bally = 60;</code>	Initial vertical coordinate for ball
<code>var maskrad;</code>	Used for mask radius

<code>var ballvx = 2;</code>	Initial ball horizontal displacement
------------------------------	--------------------------------------

Code Line	Description
<code>var ballvy = 4;</code>	Initial ball vertical displacement
<code>var v;</code>	Will hold video element
<code>function restart() {</code>	Function header for restart
<code>v.currentTime=0;</code>	Reset place in video to the start
<code>v.play();</code>	Play video
<code>}</code>	Close restart function
<code>function init(){</code>	Function header for init
<code>canvas1 = document.getElementById('canvas');</code>	Set reference for canvas
<code>ctx = canvas1.getContext('2d');</code>	Set reference for canvas context

canvas1.width = window.innerWidth;	Set canvas width to match current window width
cwidth = canvas1.width;	Set variable
canvas1.height = window.innerHeight;	Set canvas height to match current window height
cheight = canvas1.height;	Set variable
v = document.getElementById("vid");	Set reference to video element
v.addEventListener("ended",restart,false);	Set up event handling when video ends; done because loop attribute setting in element header does not work in Firefox browser
v.width = Math.min(v.videoWidth/3,.5*cwidth);	Set video width
v.height = Math.min(v.videoHeight/3,.5*cheight);	Set video height
videow = v.width;	Set variable
videoh = v.height;	Set variable

Code Line	Description
<code>ballrad = Math.min(50,.5*videow,.5*videoh);</code>	Modify ballrad if there is a very small video
<code>maskrad = .4*Math.min(videow,videoh);</code>	Set maskrad based on video dimensions
<code>ctx.lineWidth = ballrad;</code>	Set line width for drawing the box
<code>ctx.strokeStyle = "rgb(200,0,50)";</code>	Set color to reddish
<code>ctx.fillStyle="white";</code>	Set fill style for mask to be white
<code>v.play();</code>	Start video
<code>setInterval(drawscene,50);</code>	Set up timed event

}	Close init function
function drawscene(){	Function header for drawscene
ctx.clearRect(0,0,cwidth,cheight);	Erase canvas
moveandcheck();	Check if next move is at a wall, and if so, adjust displacements and position; otherwise, just make the move
ctx.drawImage(v,ballx, bally, videow,videoh);	Draw image from video at indicated position
ctx.beginPath();	Start the path for the top half of the mask
ctx.moveTo(ballx,bally);	Move to starting point
ctx.lineTo(ballx+videow,bally);	Move over horizontally
ctx.lineTo(ballx+videow,bally+.5*videoh);	Move down to halfway

<code>ctx.lineTo(ballx+.5*videow+maskrad, bally+.5*videoh);</code>	Move in to the start of where the opening will be
<code>ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0, Math.PI,true);</code>	Make semicircular arc

Code Line	Description
<code>ctx.lineTo(ballx,bally+.5*videoh);</code>	Move to the left
<code>ctx.lineTo(ballx,bally);</code>	Move to start
<code>ctx.fill();</code>	Fill in the white top of the mask
<code>ctx.moveTo(ballx,bally+.5*videoh);</code>	Move to start the bottom of the mask; move to point midway down on the left
<code>ctx.lineTo(ballx,bally+videoh);</code>	Move down to the lower left
<code>ctx.lineTo(ballx+videow,bally+videoh);</code>	Move over to the right corner
<code>ctx.lineTo(ballx+videow,bally+.5*videoh);</code>	

	Move up to the middle on the right
<code>ctx.lineTo(ballx+.5*videow+maskrad,bally+.5*videoh);</code>	Move in to the start of the hole in the mask
<code>ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0,Math.PI,false);</code>	Make semicircular arc
<code>ctx.lineTo(ballx,bally+.5*videoh);</code>	Move to the right
<code>ctx.fill();</code>	Fill in the white bottom of the mask
<code>ctx.strokeRect(0,0,cwidth,cheight);</code>	Draw the box
<code>}</code>	Close drawscene function
<code>function moveandcheck() {</code>	Header for moveandcheck function
<code>var nballx = ballx + ballvx+.5*videow;</code>	Set up trial values for x
<code>var nbally = bally +ballvy+.5*videoh;</code>	Set up trial values for y
<code>if (nballx > cwidth) {</code>	Compare to right wall, on a hit
	Change sign of the horizontal

ballvx =-ballvx;	displacement
nballx = cwidth;	Set trial value to be exactly at the right wall

Code Line	Description
}	Close clause
if (nballx < 0) {	Compare to left wall, on a hit
nballx = 0;	Set trial value to be exactly at the left wall
ballvx = -ballvx;	Change sign of the horizontal displacement
}	Close clause
if (nbally > cheight) {	Compare to bottom wall, on a hit
nbally = cheight;	Set trial value to exact height
bally =-bally;	Change the sign of the vertical displacement
}	Close clause

if (nbally < 0) {	Compare to top wall on a hit
 nbally = 0;	Change trial value to be exactly at the top wall
 ballvy = -ballvy;	Change the sign of the vertical displacement
}	Close clause
 ballx = nbally-.5*videow;	Set ballx using trial value, and offset to be the upper-left corner, not the center
 bally = nbally-.5*videoh;	Set bally using the trial value, and offset to be the upper-left corner, not the center
}	Close moveandcheck function
function reverse() {	Function header for the button action
 ballvx = -ballvx;	Change sign of horizontal displacement

Code Line	Description
 ballvy = -ballvy;	Change sign of vertical displacement
}	Close reverse function

</script>	Closing script tag
</head>	Closing head tag
<body onLoad="init();">	Opening body tag; set up call to init
<video id="vid" loop="loop" preload="auto">	Video element header
<source src="joshuahomerun.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>	Source for the mp4 video
<source src="joshuahomerun.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'>	Source for the WEBM video
<source src="joshuahomerun.theora.ogv" type='video/ogg; codecs="theora, vorbis"'>	Source for the OGG video
Your browser does not accept the video tag.	Message for noncompliant browsers
</video>	Close video tag
<button id="revbtn" onClick="reverse();">Reverse </button> 	Button for viewer to reverse direction
<canvas id="canvas" >	Opening canvas tag
This browser doesn't support the HTML5 canvas element.	Message for noncompliant browsers

</canvas>	Closing canvas tag
</body>	Closing body tag
</html>	Closing html tag

The second version of this application moves the video element as opposed to drawing the current frame of the video on the canvas. My research indicates that this may use less computer resources when it is executing. All versions have much in common, and I will point this out by only commenting on the lines that are different.

Table 3-3. Complete Code for the VideobounceE Program

Code Line	Description
<!DOCTYPE html>	
<html>	
<head>	
<title>Video bounce</title>	
<meta charset="UTF-8">	

<style>	
#vid {position:absolute; display:none; z-index: 1;	Need to set positioning and z-index because display setting will be changed to make element visible
}	End directive
#canvas {position:absolute; z-index:10; top:0px; left:0px;}	This will be on top of video and under button
#revbtn {position:absolute; z-index:20;}	
</style>	
<script type="text/javascript">	
var ctx;	
var cwidth ;	
var cheight ;	
var ballrad = 50;	
var ballx = 80;	Starting point is arbitrary
var bally = 80;	Starting point is arbitrary

<code>var maskrad;</code>	
<code>var ballvx = 2;</code>	

<code>window.scrollTo(0,0);</code>	
<code>};</code>	
<code>v = document.getElementById("vid");</code>	
<code>v.addEventListener("ended",restart,false);</code>	
<code>v.width= Math.min(v.videoWidth/3,.5*cwidth);</code>	
<code>v.height= Math.min(v.videoHeight/3,.5*cheight);</code>	
<code>videow = v.width;</code>	
<code>videoh = v.height;</code>	

Code Line	Description
<code>ballrad = Math.min(50,.5*videow,.5*videoh);</code>	
<code>maskrad = .4*Math.min(videow,videoh);</code>	
<code>ctx.lineWidth = ballrad;</code>	
<code>ctx.strokeStyle = "rgb(200,0,50)";</code>	
<code>ctx.fillStyle= "white";</code>	
<code>v.style.left = String(ballx)+"px";</code>	
<code>v.style.top = String(bally)+"px";</code>	
<code>v.play();</code>	

<code>v.style.display = "block";</code>	Make video element visible
<code>setInterval(drawscene,50);</code>	
<code>}</code>	
<code>function drawscene(){</code>	
<code>ctx.clearRect(0,0,cwidth,cheight);</code>	
<code>moveandcheck();</code>	
<code>v.style.left = String(ballx)+"px";</code>	Position video horizontally
<code>v.style.top = String(bally)+"px";</code>	Position video vertically
<code>ctx.beginPath();</code>	
<code>ctx.moveTo(ballx,bally);</code>	
<code>ctx.lineTo(ballx+videow,bally);</code>	
<code>ctx.lineTo(ballx+videow,bally+.5*videoh);</code>	
<code>ctx.lineTo(ballx+.5*videow+maskrad,</code> <code>bally+.5*videoh);</code>	

Code Line	Description
<code>ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0,</code> <code>Math.PI,true);</code>	

ctx.lineTo(ballx,bally+.5*videoh);	
ctx.lineTo(ballx,bally);	
ctx.fill();	
ctx.moveTo(ballx,bally+.5*videoh);	
ctx.lineTo(ballx,bally+videoh);	
ctx.lineTo(ballx+videow,bally+videoh);	
ctx.lineTo(ballx+videow,bally+.5*videoh);	
ctx.lineTo(ballx+.5*videow+maskrad,bally+.5*videoh);	
ctx.arc(ballx+.5*videow,bally+.5*videoh,maskrad,0, Math.PI,false);	
ctx.lineTo(ballx,bally+.5*videoh);	
ctx.fill();	
ctx.strokeRect(0,0,cwidth,cheight); // box	

<code>}</code>	
<code>function moveandcheck() {</code>	
<code> var nballx = ballx + ballvx;</code>	Trial value
<code> var nbally = bally +ballvy;</code>	Trial value
<code> if ((nballx+videoow) > cwidth) {</code>	Add total width and compare
<code> ballvx =-ballvx;</code>	Change sign of horizontal displacement
<code> nballx = cwidth-videoow;</code>	Set to exact position

Code Line	Description
<code>}</code>	
<code>if (nballx < 0) {</code>	
<code> nballx = 0;</code>	
<code> ballvx = -ballvx;</code>	
<code>}</code>	

if ((nbally+videoh) > cheight) {	Compare total length
nbally = cheight-videoh;	Set to exact position
ballvy = -ballvy;	Change sign of vertical displacement
}	
if (nbally < 0) {	
nbally = 0;	
ballvy = -ballvy;	
}	
ballx = nbally;	Set to trial position, possibly adjusted
bally = nbally;	Set to trial position, possibly adjusted
}	
function reverse() {	
ballvx = -ballvx;	
ballvy = -ballvy;	
}	
</script>	
</head>	

Code Line	Description
-----------	-------------

<body onLoad="init(); ">	
<video id="vid" loop="loop" preload="auto">	
<source src="joshuahomerun.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'>	
<source src="joshuahomerun.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>	
<source src="joshuahomerun.theora.ogv" type='video/ogg; codecs="theora, vorbis"'>	
Your browser does not accept the video tag.	
</video>	
<button id="revbtn" onClick="reverse(); ">Reverse </button> 	
<canvas id="canvas" >	
This browser doesn't support the HTML5 canvas element.	
</canvas>	
</body>	

</html>	
---------	--

I made the **trajectory** function by modifying the **drawscene** to videobounceC. Since I wanted the circle to be similar in size to the masked video clip, I added an **alert** statement temporarily to the videobounceC function after the video width and height were set, and ran the program using those values:

```
v.width = Math.min(v.videoWidth/3,.5*cwidth);
```

```
v.height = Math.min(v.videoHeight/3,.5*cheight);
```

```
alert("width "+v.width+" height "+v.height);
```

I then used the values, 106 and 80, to be the **videow** and **videoh** values in the trajectory program. The complete code, with the changed lines annotated, is shown in Table 3-4. Note that the main difference between this program and the first two is the missing lines.

Table 3-4. Complete code for VideobounceTrajectory Program

Code Line	Description
<!DOCTYPE html>	
<html>	
<head>	

<title>Video bounce</title>	
<meta charset="UTF-8">	
<style>	
#canvas {position:absolute; z-index:10; top:0px; left:0px;}	
#revbtn {position:absolute; z-index:20; }	
</style>	
<script type="text/javascript">	
var ctx;	
var cwidth ;	
var cheight ;	
var ballrad = 50;	
var ballx = 50;	
var bally = 60;	
var ballvx = 2;	
var ballvy = 4;	
function init(){	
canvas1 = document.getElementById('canvas');	

Code Line	Description
<code>ctx = canvas1.getContext('2d');</code>	
<code>canvas1.width = window.innerWidth;</code>	
<code>cwidth = canvas1.width;</code>	
<code>canvas1.height = window.innerHeight;</code>	
<code>cheight = canvas1.height;</code>	
<code>videow= Math.min(106,.5*cwidth);</code>	Use values of actual video width
<code>videooh = Math.min(80,.5*cheight);</code>	Use values of actual video height
<code>ballrad = Math.min(50,.5*videow,.5*videooh);</code>	
<code>maskrad = .4*Math.min(videow,videooh);</code>	
<code>ctx.lineWidth = ballrad;</code>	
<code>ctx.fillStyle= "white";</code>	
<code>ctx.strokeStyle ="rgb(200,0,50)";</code>	
<code>ctx.strokeRect(0,0,cwidth,cheight);</code>	
<code>setInterval(drawscene,50);</code>	
{	
<code>function drawscene(){</code>	

moveandcheck();	
ctx.beginPath();	Begin path for circle
ctx.moveTo(ballx+.5*videoW+maskrad, bally+.5*videoH);	Move to point on circle at right
ctx.arc(ballx+.5*videoW,bally+.5*videoH,maskrad,0, 2*Math.PI,true);	Draw circle
ctx.lineWidth=1;	Set line width

Code Line	Description
ctx.fill();	Fill in (this will be white)
ctx.stroke();	Set red stroke (outline)
ctx.lineWidth= ballrad;	Set line width for the box
ctx.strokeRect(0,0,cwidth,cheight);	Draw the box
}	
function moveandcheck() {	
var nballx = ballx + ballvx+.5*videoW;	
var nbally = bally +ballvy+.5*videoH;	
if (nballx > cwidth) {	
 ballvx =-ballvx;	

nballx = cwidth;	
}	
if (nballx < 0) {	
nballx = 0;	
ballvx = -ballvx;	
}	
if (nbally > cheight) {	
nbally = cheight;	
ballvy =-ballvy;	
}	
if (nbally < 0) {	
nbally = 0;	

Code Line	Description
ballvy = -ballvy;	
}	
ballx = nballx-.5*videow;	
bally = nbally-.5*videoh;	

```
}

function reverse() {

    ballvx = -ballvx;

    ballvy = -ballvy;

}

</script>

</head>

<body onLoad="init();">

<button id="revbtn"
        onClick="reverse();">Reverse </button><br/>

<canvas id="canvas" >

This browser doesn't support the HTML5 canvas
element.

</canvas>

</body>

</html>
```

Making the Application Your Own

The first way to make this application your own is to use your own video. You do need to find something that is acceptable when displayed as a small circle. As mentioned earlier, you need to produce versions using the different video codecs. A next step is adding other user interface actions, including changing the horizontal and vertical speeds, as was done in the bouncing ball projects in *The Essential Guide to HTML5*. Another set of enhancements would be to add video controls. Video controls can be part of the video element, but I don't think that would work for a video clip that needs to be small and is moving! However, you could implement your own controls with buttons modeled after the Reverse button. For example, the statement

```
v.pause();
```

does pause the video.

The attribute **v.currentTime** can be referenced or set to control the position within the video clip. You saw how the range input type works in Chapter 1, so consider building a slider input element to adjust the video.

You may decide you want to change my approach to adapting to the window dimensions. One alternative is to change the video clip dimensions to maintain the aspect ratio. Another alternative is to change the video dimensions all the time. This means that the video dimensions and the canvas

directions will be in proportion all the time. Yet another alternative, though I think this will be disconcerting, is to make reference to the window dimensions at each time interval and make changes in the canvas, and possibly the video, each time. There is an event that can be inserted into the body tag:

```
<body onresize="changedims();" ... >
```

This coding assumes that you have defined a function named **changedims** that includes some of the statements in the current **init** function to extract the **window.innerWidth** and **window.innerHeight** attributes to set the dimensions of the canvas and the video.

More generally, the objective of this chapter is to show you ways to incorporate video into your projects in a dynamic fashion, both in terms of position on the screen and timing. In particular, it is possible to combine playing of video with drawings on a canvas for exciting effects.

Screen savers exist in which the screen is filled up by a bouncing object similar to the trajectory program. You can change the **drawscene** function to produce different shapes. Also, as I mentioned before, you can apply the techniques explained in *The Essential Guide to HTML5* to provide actions by the viewer. You can refer to Chapter 1 in this book for the use of a range input (slider). Yet another possibility is to provide the viewer a way to change the color of the circle (or other shape you design) using the input type of color.

The Opera browser provides a color-picker option.

Testing and Uploading the Application

As has been mentioned, but is worth repeating, you need to acquire a suitable video clip. At the time of writing this book, you then need to use a program such as Miro to produce the WEBM, mp4, and OGG versions because browsers recognize different video encodings (codecs). This situation may change. Again, if you are content with implementing this for just one browser, you can check which video encoding works for that browser and just prepare one video file. The video files and the **html** file need to be in the same folder on your computer and in the same folder on your server if and when you upload this application to your server account. Alternatively, you can use a complete web address or the correct relative address in the source elements.

Summary

In this chapter, you learned different ways to manipulate video. These included the following:

- Drawing the current frame of video as an image onto a canvas
- Repositioning of a video element on the screen by changing the **left** and **top** style attributes
- Using style directives to layer a video, a canvas, and a button
- Creating a moving mask on a canvas
- Acquiring information on the dimensions of the window to adapt an application to different situations

The next chapter will show you how to use the Google Maps Application Programming Interface (API) in an HTML5 project. The project will involve using a canvas and changing the z-index so that the canvas is alternatively under and over the material produced by Google Maps.

CHAPTER 4

Map Maker: Combining Google Maps and the Canvas

In this chapter, you will learn how to do the following:

- Use the Google Maps API to display a map at a specific location
- Draw graphics on a canvas using transparency (also known as the alpha or opacity level) and a customized cursor icon
- Provide a graphical user interface (GUI) to your users by combining the use of Google Maps and HTML5 features by managing the events and the z-index levels
- Calculate the distance between two geographical locations

Introduction

The project for this chapter is an application involving a geographic map. Many applications today involve the use of an Application Programming Interface (API) provided by another person or organization. This chapter will be an introduction to the use of the Google Maps API, and is the first of three chapters using the Google Maps JavaScript Version 3 API. Figure 4-1 shows the opening screen.

Base location (small red x)

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE



Figure 4-1. Opening screen of map spotlight project

Notice the small red (hand-drawn) x located just above and to the left of SoHo. When deciding on map markers, you face a trade-off. A smaller marker is more difficult to see. A larger and/or more intricate marker is easier to see but blocks more of the map or distracts from the map. The x marks a neighborhood in lower Manhattan. It is the address of the friends of ED publishers. For this program, it is the initial base location. The base location is used to calculate distances.

Moving the mouse over the map is shown in Figure 4-2.

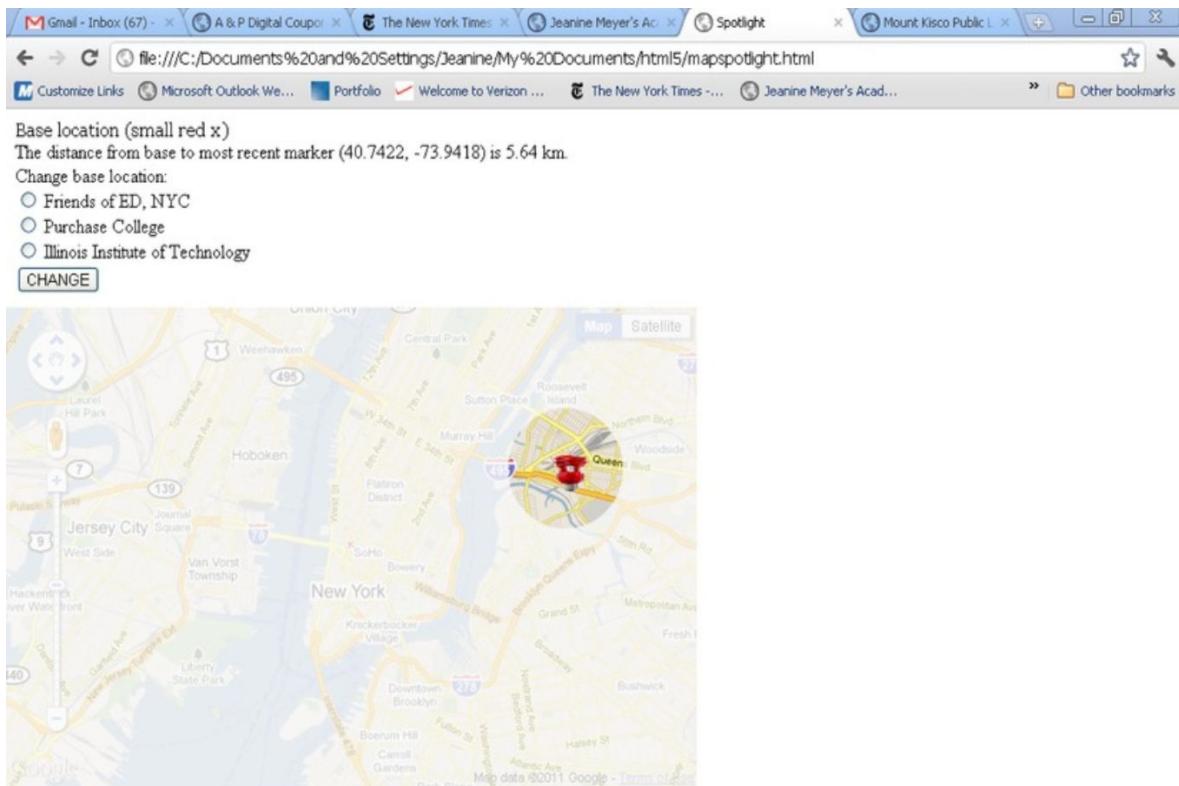


Figure 4-2. Shadow/spotlight over map

Notice the shadow and spotlight combination now on the map. Most of the map is covered by a semitransparent shadow. You need to trust me that this screenshot was taken when I had moved the mouse over the map. There is a circle around the mouse position in which the original map shows through. The cursor is not the standard, default cursor but one I created using a small image representing a compact fluorescent lightbulb.

The text on the screen shows the distance from the base to the last spot on the map I clicked to be 5.64 kilometers. The marker for all such locations is a hand-drawn x. The latitude and longitude of this location is indicated in parentheses.

The general GUI features provided by Google Maps are available to the users of

this project. This includes a vertical slider that controls the scale of the presentation. Figure 4-3 demonstrates the result of using that slider to zoom in. It is possible to zoom in even further.

Base location (small red x)

The distance from base to most recent marker (40.7422, -73.9418) is 5.64 km.

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE



Figure 4-3. Zoomed in to street level

It also is possible to pan the map by clicking the hand in the upper-left corner and then virtually grabbing the mouse by pressing down on the mouse button and pulling. Figure 4-4 shows the effects of zooming out and moving north. The screen shows the shadow/spotlight again.

Base location (small red x)
The distance from base to most recent marker (41.2365, -73.6977) is 62.39 km.
Change base location:
 Friends of ED, NYC
 Purchase College
 Illinois Institute of Technology
CHANGE

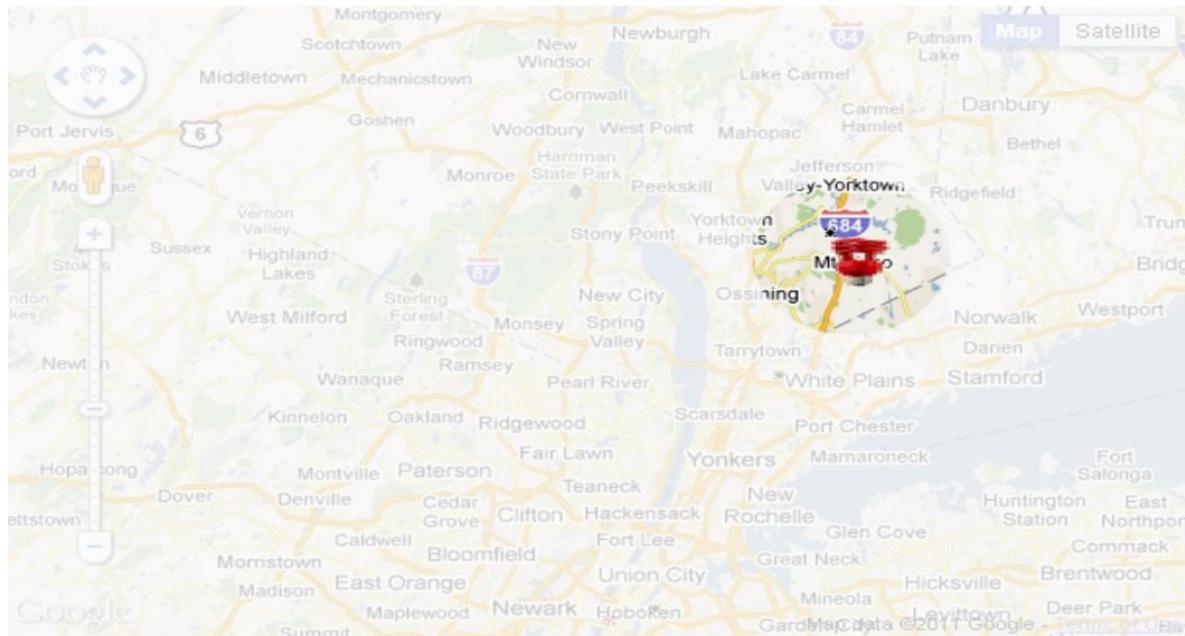
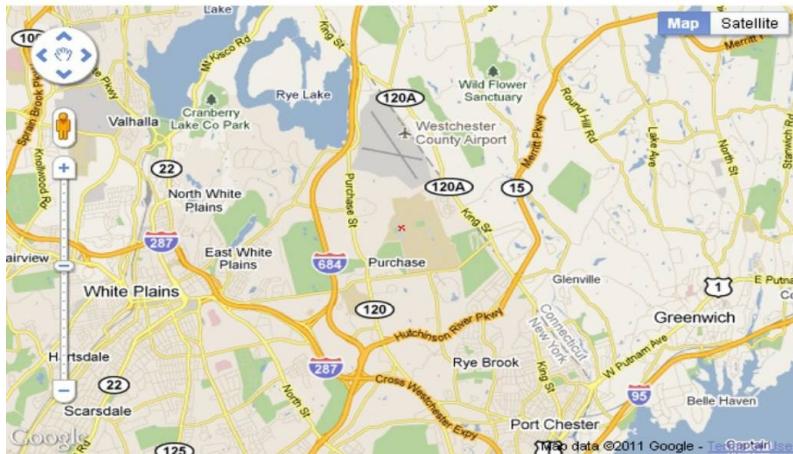


Figure 4-4. Zooming out and moving north

Base location (small red x) is Purchase College/SUNY
Change base location:
 Friends of ED, NYC
 Purchase College
 Illinois Institute of Technology
[CHANGE](#)



The program provides a way to change the base location. There are three choices: the location of the publisher, friends of ED, in New York City; Purchase College (where I teach), located in Purchase, New York, which is north of New York City; and Illinois

Institute of Technology (where my son teaches), located in Chicago, Illinois. The interface for making this selection is a set of radio buttons—only one button can be selected at a time—and a button labeled CHANGE to be clicked when the user/viewer/visitor decides to make a change. Figure 4-5 shows the results of making a change to Purchase College. Notice the hand-drawn red x marking the base location and the text at the top of the page indicating the new base location by name.

Figure 4-5. Purchase College new base location

Next, I switch to Illinois Institute of Technology in Chicago, Illinois, as the base by clicking the third radio button. The result is shown in Figure 4-6.

Base location (small red x) is Illinois Institute of Technology

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE

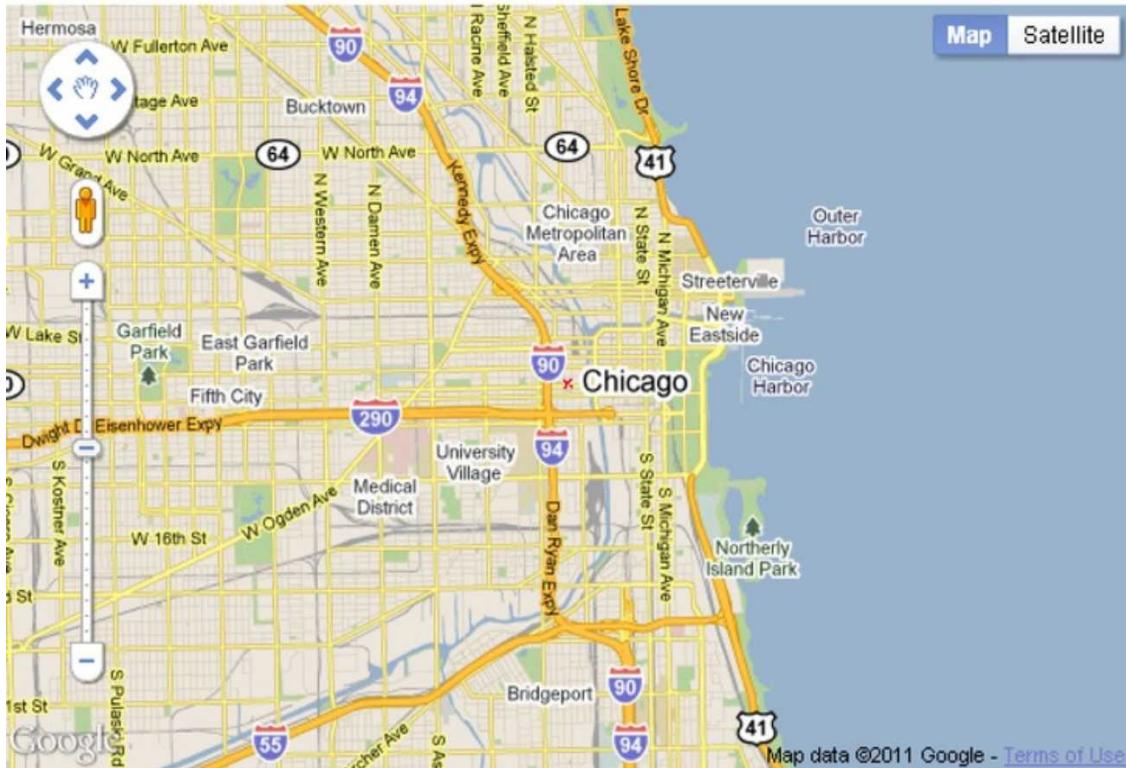


Figure 4-6. Base at Illinois Institute of Technology in Chicago

Again, notice the small red x indicating the base location and the text at the top of the screen with the name of the new base location.

The location of each base is determined by the latitude and longitude values for each of the three values that I have determined. My code is not “asking” Google Maps to find these locations by name. You may get different results if you type the terms “Purchase College, NY” and “Illinois Institute of Technology” into Google or Google Maps. To make this application your own, you would decide on a set of base locations and look up the latitude and longitude values. I will suggest ways to do this in the next section.

Just in case you are curious, zooming out to the farthest out position on the zoom/scale produces what is shown in Figure 4-7. This projection exhibits what is called the Greenland problem. Greenland is not bigger than Africa, but actually about 1/14 times the size.

Base location (small red x)

The distance from base to most recent marker (40.6855, -74.0935) is 8.7 km.

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE

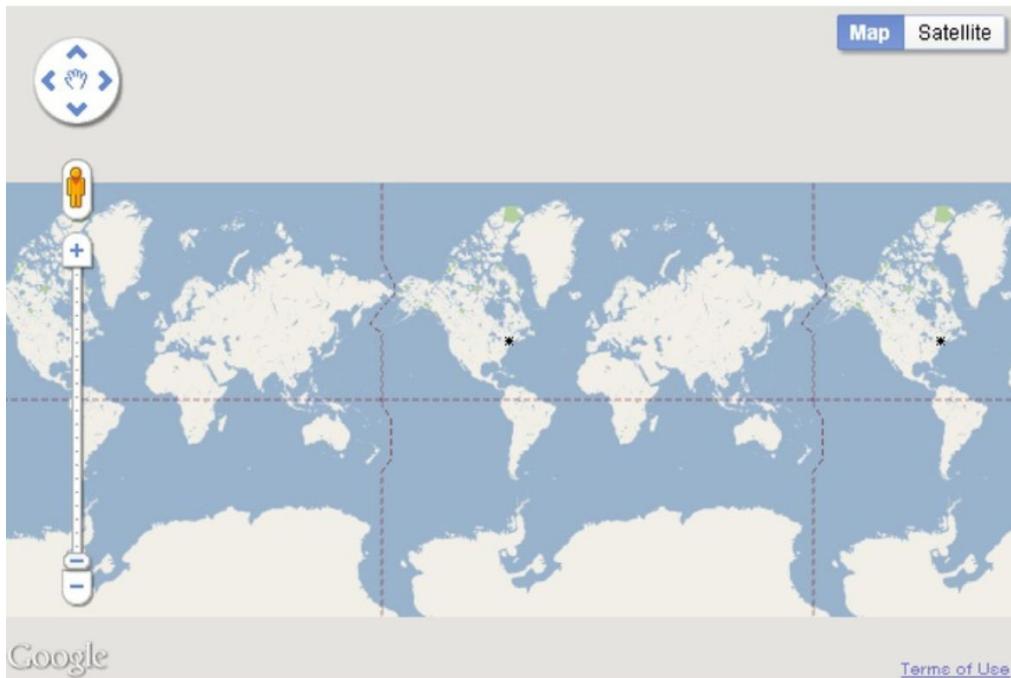


Figure 4-7. Farthest-out view of map

Figure 4-8 shows the map at close to the closest-in limit. The map has also been changed to the satellite view using the buttons in the upper-right corner.

Base location (small red x) is Friends of ED, NYC

The distance from base to most recent marker (40.6855, -74.0935) is 8.7 km.

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE

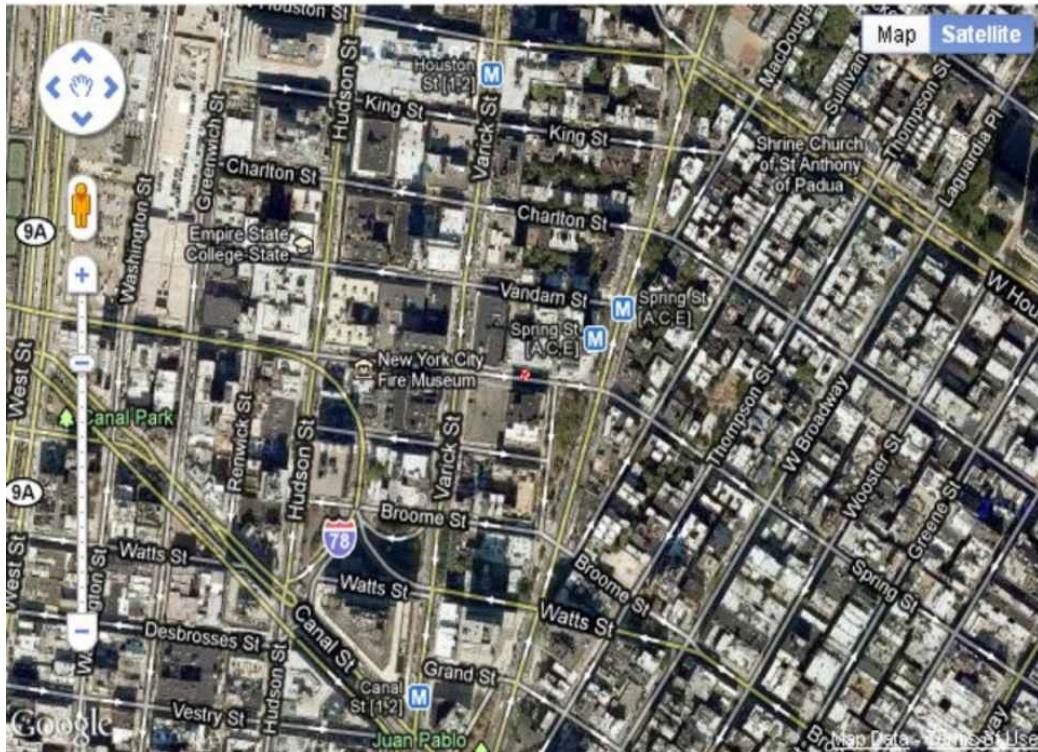


Figure 4-8. Zoomed in to where city blocks can be detected

Notice that the slider on the left is about four notches above the closest setting. Lastly, Figure 4-9 shows the map zoomed in to the limit. This is essentially at the building level, at least for Manhattan.

Base location (small red x) is Friends of ED, NYC

The distance from base to most recent marker (40.7257, -74.0049) is 0.01 km.

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE



Figure 4-9. Zoomed in all the way

By using the interface to zoom out and pan and zoom in again, I can determine the distance from any of the base locations to any other location in the world! I also can use this application to determine latitude and longitude values of any location. You need to know the latitude and longitude for changing or adding to the list of base locations and for determining locations for the project in Chapter 5. I review latitude and longitude in the next section.

Google Maps by itself is an extremely useful application. This chapter and the next two demonstrate how to bring that functionality into your own application. That is, we combine the general facilities of Google Maps with anything, or almost anything, we can develop using HTML5 and JavaScript.

Latitude & Longitude and Other Critical Requirements

The most fundamental requirement for this project, and the ones in the next two chapters, is an understanding of the coordinate system for geography. Just as a coordinate system is required for specifying points on a canvas or positions on the screen, it is necessary to use a system for places on planet earth. The latitude and longitude system has been developed and standardized over the last several hundred years. The values are angles, with latitude indicating degrees from the equator and longitude indicating degrees from the Greenwich prime meridian in the United Kingdom. The latter is an arbitrary choice that became standard in the late 1800s. There is a northern hemisphere bias here: latitude values go from 0 degrees at the equator to 90 degrees at the North Pole and –90 degrees at the South Pole. Similarly, longitude values are positive going east from the Greenwich prime meridian and negative going west. Latitudes are parallel to the equator and longitudes are perpendicular. Latitudes are often called parallels and typically appear as horizontal lines, and longitudes are called meridians and typically appear as verticals. This orientation is arbitrary, but fairly solidly established.

I will use decimal values, which is the default displayed in Google Maps, but you will see combinations of degree, minute (1/60 of a degree), and second (1/60 of a minute). It is not necessary that you memorize latitude longitude

values, but it is beneficial to develop some intuitive sense of the system. You can do this by doing what I call “going both ways.” First, identify and compare latitude longitude values for places you know, and second, pick values and see what they are. For example, the base values for my version of the project are as follows:

- [40.725592, -74.00495, “friends of ED, NYC”]
- [41.04796, -73.70539, “Purchase College/SUNY”]
- [41.878928, -87.641926, “Illinois Institute of Technology”]

The first thing to notice is that the latitude values are fairly close and the longitude values are negative and not quite so close. The friends of ED office in New York City is within 1 degree of latitude and 1 degree of longitude of Purchase College. The distance according to Google Maps is 27.4 miles. The longitude value for Illinois Institute of Technology is more negative, indicating that it’s more westerly than the two New York State locations. This all makes sense, but you need to take the time to think it through.

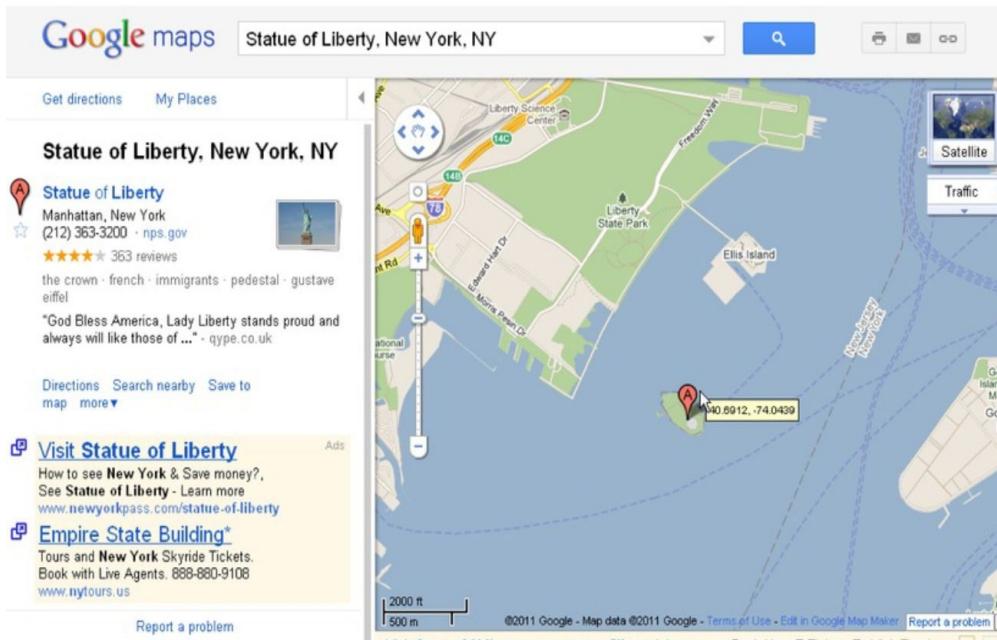
There are many ways to find the latitude and longitude of a specific location. You can use Google Maps as follows:

1. Invoke Google Maps (go to www.google.com and click Maps or go to

<http://maps.google.com>.

2. At the upper right, click the gear icon for a drop-down menu. Click the Maps Labs option. A window will appear titled Google Maps Labs. Scroll down to the LatLng Marker option and click the circle next to Enable. If you sign in, all settings will remain in force the next time you sign in. Save and close the window.
3. Type in the location you are interested in into the location field.
4. Right/Ctrl+click the location to get a drop-down menu, as shown in Figure 4-10.
5. Click Drop LatLng marker to get the result shown in Figure 4-11.

Figure 4-10. Getting latitude longitude values in Google Maps



After choosing

the Drop LatLng Marker option, you will see the latitude and longitude values in a small box, as shown in Figure 4-11.

Figure 4-11. Box showing latitude and longitude

Another option is to use Wolfram Alpha (www.wolframalpha.com), as shown in Figure 4-12, which provides a way to determine latitude and longitude values as well as many other things.

The screenshot shows a web browser window with the URL www.wolframalpha.com/input/?i=latitude+longitude%3A+Statue+of+Liberty. The browser's address bar and various tabs are visible at the top. Below the address bar, the Wolfram Alpha logo and navigation links for HOME, EXAMPLES, PRODUCTS, BLOG, and ABOUT are present. The main content area displays the query "latitude longitude: Statue of Liberty" in a search bar. Below the search bar, a note says "Assuming 'Statue of Liberty' is a structure | Use as a park instead". The input interpretation section shows "Statue of Liberty coordinates" and the result "40° 41' 21"N, 74° 2' 40"W". At the bottom, there are links for "Computed by Wolfram Mathematica", "Source information >", and "Download as: PDF | Live Mathematica".

Figure 4-12. Results of query on Wolfram Alpha

Notice the format of the results. This is the degree/minute/second format, with N for north and W for west. When I click the “Show decimal” button, the program displays what is shown in Figure 4-13.

The screenshot shows the Wolfram Alpha interface. At the top, the logo 'WolframAlpha™ computational... knowledge engine' is displayed. Below it, a search bar contains the query 'latitude longitude: Statue of Liberty'. To the right of the search bar are two buttons: a blue one with a magnifying glass icon and a red one with a square icon. Below the search bar, there are 'Examples' and 'Random' buttons. A note below the search bar says 'Assuming "Statue of Liberty" is a structure | Use as a park instead'. In the main content area, under 'Input interpretation', it shows 'Statue of Liberty coordinates'. The 'Result' section displays '40.69°N, 74.04°W'. To the right of the result is a 'Show DMS' button. At the bottom of the interface, there are links for 'Computed by Wolfram Mathematica', 'Source information >', and 'Download as: PDF | Live Mathematica'.

Figure 4-13. Decimal results for query to Wolfram Alpha

Notice that the longitude still appears with W for West as opposed to the negative value given by Google Maps.

Doing what I call “going in the opposite direction,” you can put latitude and

longitude values into Google Maps. Figure 4-14 shows the results of putting in 0.0 and 0.0. It is a point in the ocean south of Ghana. This is a point on the equator *and* on the Greenwich prime meridian.

Figure 4-14. The equator at the Greenwich prime meridian

I tried to find a place in England on the Greenwich prime meridian and produced the result shown in Figure 4-15 when guessing at the latitude of 52.0 degrees.

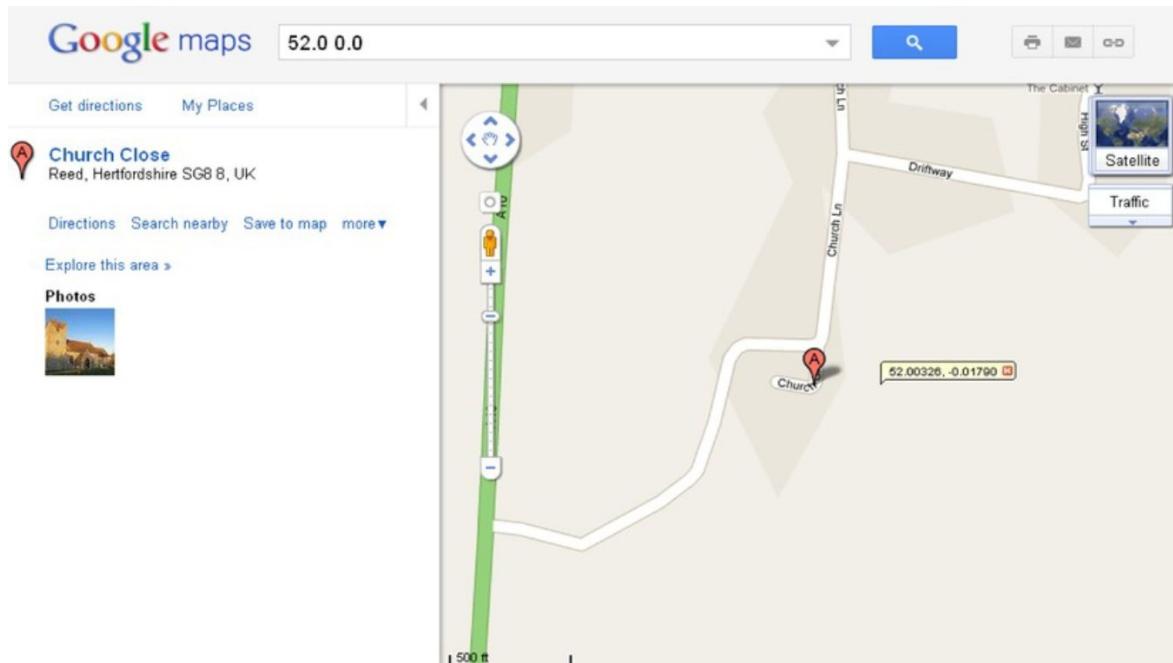


Figure 4-15. Results near a place on the Greenwich prime meridian

The A marker indicates the closest place in the Google database to the requested location. I used the Drop LatLng marker option to reveal the exact latitude and longitude values.

The critical requirements for this project start off with the task of bringing Google Maps into a HTML5 application using specified latitude and longitude values. An additional requirement is producing the shadow/spotlight combination on top of the map to track the movements of the mouse. I also require a change from the default cursor for the mouse to something of my own choosing.

Next, I added a requirement to drop markers on the map, but again, with

graphical icons that I picked, not the upside-down teardrop that is standard in Google Maps. The teardrop marker is nice enough, but my design objective was to be different to show you how to incorporate your own creativity into an application.

Beyond the graphics, I wanted the users to be able to make use of the Google Maps devices and any GUI features I built using HTML5. This all required managing events set up by the Google Maps API and events set up using HTML5 JavaScript. The responses to events that I wanted to make the user interface included the following:

- Tracking mouse movement with the shadow/spotlight graphic
- Responding to a click by placing an x on the map
- Retaining the same response to the Google Maps interface (slider, panning buttons, panning by grabbing the map)
- Treating the radio buttons and CHANGE button in the appropriate manner

Google Maps provides a way to determine distances between locations. Since I wanted to set up this project to work in terms of the base location, I needed a way to calculate distances directly.

These are the critical requirements for the map spotlight project. Now I will explain the HTML5 features I used to build the project. The objective is to use

Google Maps features and JavaScript features, including events, and not let them interfere with each other. You can use what you learn for this and other projects.

HTML5, CSS, and JavaScript Features

The challenges for the map-maker project are bringing in the Google Map and then using the map and canvas and buttons together in terms of appearance and in the operation of the GUI. I'll describe the basic Google Maps API and then explain how HTML5 features provide the partial masking and the event handling.

Google Maps API

The Google Maps JavaScript API Version 3 Basics has excellent documentation located at

<http://code.google.com/apis/maps/documentation/javascript/basics.html>.

You do not need to refer to it right now, but it will help you if and when you decide to build your own project. It will be especially helpful in producing applications for mobile devices.

Most APIs are presented as a collection of related objects, each object having attributes (also known as properties) and methods. The API also may include events and a method for setting up the event. This is the situation with the Google Maps API. The important objects are **Map**, **LatLng**, and **Marker**. The method to set up an event is **addListener**, and this can be used to set up a response to clicking a map.

The Google Maps API is brought into your HTML5 document with a script element:

```
<script type="text/javascript" charset="UTF-8"
src="http://maps.google.com/maps/api/js?
sensor=false"></script>
```

I will discuss the script tag again, specifically the **sensor=false** setting, in Chapter 6.

The next step—and this could be all you need if all you want is to bring in a Google Map—is to set up a call to the **Map** constructor method. Pseudocode for this is

```
map = new google.maps.Map(place you are going to put the map, associative array with options);
```

Note that there is no harm in making the variable have the name **map**.

Let's take up the two parameters one at a time. The place to put the map could be a div defined in the body of the HTML document. However, I chose to create the div dynamically. I did this using code in an **init** function invoked in the usual way, by setting the **onLoad** attribute in the body statement. I also wrote code to create a canvas element inside the div. The code is

```
candiv = document.createElement("div");

candiv.innerHTML = ("<canvas id='canvas'  
width='600' height ='400'>No canvas </canvas>");

document.body.appendChild(candiv);

can = document.getElementById("canvas");
pl = document.getElementById("place");
ctx = can.getContext("2d");
```

can, **pl**, and **ctx** are global variables, each available for use by other functions.

Note Though I try to use the language “bring access to Google Maps into the HTML document,” I am guilty of describing a function that “makes” a map. The Google Maps connection is a dynamic one in which Google Maps creates what are termed “tiles to be displayed.”

The second parameter to the **Map** method is an associative array. An associative array has named elements, not indexed elements. The array for the **Map** method can indicate the zoom level, the center of the map, and the map type, among other things. The zoom level can go from 0 to 18. Level 0 is what is shown in Figure 4-7. Level 18 could show buildings. The types of maps are ROADMAP, SATELLITE, HYBRID, and TERRAIN. These are indicated using constants from the Google Maps API. The center is given by a value of type **LatLng**, constructed, as you may expect, using decimal numbers representing latitude and longitude values. The use of an associative array means that we don’t have to follow a fixed order for parameters, and default settings will be applied to any parameter we omit.

The start of my **makemap** function follows. The function is called with two numbers indicating the latitude and longitude on which to center the map. My code constructs a **LatLng** object, sets up the array holding the specification for the map, and then constructs the map—that is, constructs the portal to Google Maps.

```
function makemap(mylat,mylong) { var marker;
```

```
blatlng = new google.maps.LatLng(mylat,mylong);
```

```
myOptions = {
```

```
zoom: 12,
```

```
center: blatlng,
```

```
mapTypeId: google.maps.MapTypeId.ROADMAP };
```

```
map = new google.maps.Map(document.getElementById("place"), myOptions);
```

The **Map** method constructs access to Google Maps starting with a map with the indicated options in the div with the ID **place**. The **makemap** function continues, placing a marker at the center of the map. This is done by setting up an associative array as the parameter for the **Marker** method. The icon marker will be an image I created using an image of my own design, a drawn red x.

```
marker = new google.maps.Marker({
```

```
position: blatlng,
```

```
title: "center",
```

```
icon: rxmarker,
```

```
map: map } );
```

There is one more statement in the **makemap** function, but I will explain the rest later.

Canvas Graphics

The graphic that we want to move with the mouse over the map is similar to the mask used in Chapter 3 to turn the rectangular video clip into a circular video clip. Both masks can be described as resembling a rectangular donut: a rectangle with a round hole. We draw the graphics for the shadow/spotlight using two paths, just like the mask for the video in the previous chapter. There are two distinct differences, however, between the two situations:

The exact shape of this mask varies. The outer boundary is the whole canvas, and the location of the hole is aligned with the current position of the mouse. The hole moves around.

- The color of the mask is not solid paint, but a transparent gray.

The canvas starts out on top of the Google Map. I accomplish this by writing style directives that set the z-index values:

```
canvas {position: absolute; top: 165px; left: 0px; z-index: 100;}  
#place {position: absolute; top: 165px; left: 0px; z-index: 1;}
```

The first directive refers to all canvas elements. There is only one in this HTML document. Recall that the z-axis comes out of the screen toward the viewer, so higher values are on top of lower values. Note also that we use **zIndex** in the JavaScript code and **z-index** in the CSS. The JavaScript parser

would treat the – sign as a minus operator, so the change to **zIndex** is necessary. I will need to write code that changes the **zIndex** to get the event handling that I want for this project.

Figure 4-16 shows one example of the shadow mask drawn on the canvas. The canvas is over the map in terms of the z-index, and the mask is drawn with a gray color that is transparent so the map underneath is visible.

Figure 4-16. Shadow/spotlight on one place on the map

Figure 4-17 shows another example of the shadow mask drawn on the same map. This came about because of movement of the mouse by the user.

Base location (small red x) is Friends of ED, NYC
The distance from base to most recent marker (40.7257, -74.0047) is 0.03 km.

Change base location:

- Friends of ED, NYC
- Purchase College
- Illinois Institute of Technology

CHANGE

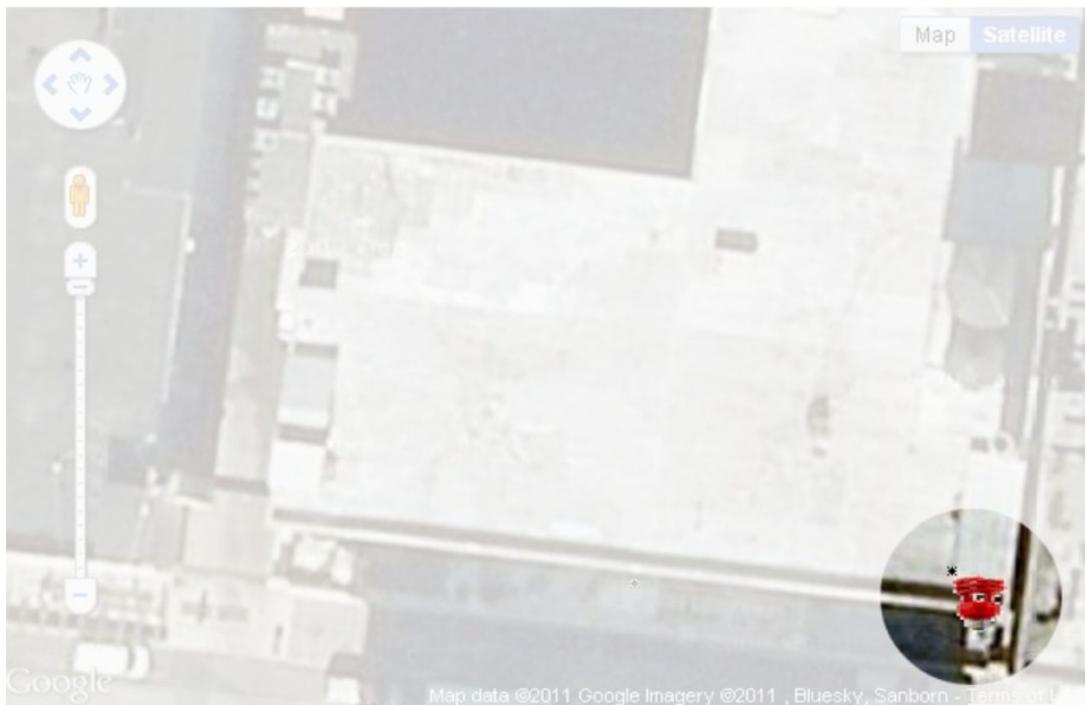


Figure 4-17. Shadow mask over another position on the map

Several topics are interlinked here. Let's assume that the variables **mx** and **my** hold the position of the mouse cursor on the canvas. I will explain how later in this chapter. The function **drawshadowmask** will draw the shadow mask. The transparent gray that is the color of the mask is defined in a variable I named **grayshadow** and constructed using the built-in function **rgba**. The **rgba** stands for red-green-blue-alpha. The alpha refers to the transparency-opacity. A value of 1 for alpha means that the color is fully opaque: solid. A value of 0 means that it is fully transparent—the color is not visible. Recall also that the red, green, and blue values go from 0 to 255, and the combination of 255, 255, and 255 would be white. This is a time for experimentation. I decided on the following setting for the gray/grayish/ghostlike shadow:

```
var grayshadow = "rgba(250,250,250,.8);"
```

The function **drawshadowmask** makes use of several variables that are constants—they never change. A schematic indicating the values is shown in Figure 4-18.

Figure 4-18. Schematic with variable values indicated for mask

The mask is drawn in two parts as was done for the mask for the bouncing video. You may look back to Figure 3-8 and Figure 3-9. The coding is similar:

```
function drawshadowmask(mx,my) {  
  
    ctx.clearRect(0,0,600,400);  
  
    ctx.fillStyle = grayshadow;  
  
    ctx.beginPath();  
  
    ctx.moveTo(canvasAx,canvasAy);  
  
    ctx.lineTo(canvasBx,canvasBy);  
  
    ctx.lineTo(canvasBx,my);  
  
    ctx.lineTo(mx+holerad,my);  
  
    ctx.arc(mx,my,holerad,0,Math.PI,true);  
  
    ctx.lineTo(canvasAx,my);  
  
    ctx.lineTo(canvasAx,canvasAy);  
  
    ctx.closePath();  
  
    ctx.fill(); ctx.beginPath();  
  
    ctx.moveTo(canvasAx,my);  
  
    ctx.lineTo(canvasDx,canvasDy);  
  
    ctx.lineTo(canvasCx,canvasCy);
```

```
ctx.lineTo(canvasBx,my);

ctx.lineTo(mx+holerad,my);

ctx.arc(mx,my,holerad,0,Math.PI,false);

ctx.lineTo(canvasAx,my);

ctx.closePath();

ctx.fill();

}
```

Now we move on to the red lightbulb.

Cursor

The cursor—the small graphic that moves on the screen when you move the mouse—can be set in the style element or in JavaScript. There are several built-in choices for the graphic (e.g., crosshair, pointer), and we also can refer to our own designs for a custom-made cursor, which is what I demonstrate in this project. I included the statement

```
can.onmousedown = function () { return false; } ;
```

in the **init** function to prevent a change to the default cursor when pressing down on the mouse. This may not be necessary since the default may not be triggered.

To change the cursor for moving the mouse to something that conveyed a spotlight, I created a picture of a red compact fluorescent lightbulb and saved it in the file **light.gif**. I then used the following statement in the function **showshadow**. The **showshadow** function has been set as the event handler for **mousemove**:

```
can.style.cursor = "url('light.gif'), pointer";
```

to indicate that JavaScript should use that address for the image for the cursor when on top of the **can** element. Furthermore, if the file 'light.gif' is not available, the statement directs JavaScript to use the built-in pointer icon. This is similar to the way that fonts can be specified with a priority listing of

choices. The variable **can** has been set to reference the canvas element. The cursor will not be used when the canvas has been pushed under the Google Map, as will be discussed in the next section.

Events

The handling of events—namely mouse events, but also events for changing the slider on the Google Map or clicking the radio buttons—seemed the most daunting when I started work on this project. However, the actual implementation turned out to be straightforward. In the **init** function, I write code to set up event handling for movement of the mouse, mouse button down, and mouse button up, all regarding the canvas element:

```
can.onmousedown = function () { return false; } ;  
  
can.addEventListener('mousemove',showshadow,true);  
  
can.addEventListener('mousedown',pushcanvasunder,true);  
  
can.addEventListener("mouseout",clearshadow,true);
```

The **true** value for the third parameter indicates that this event is to *bubble*, meaning that it is to signal other listeners. However, more work was needed to achieve the event handling I wanted for this project. I will explain the three functions and then go on to describe one more event.

The **showshadow** function, as indicated previously, calls the **drawshadowmask** function. I could have combined these two functions, but dividing tasks into smaller tasks generally is a good practice. The **showshadow** function determines the mouse position, makes an adjustment so the lightbulb base is at the center of the spotlight, and then makes the call to **drawshadowmask**:

```
function showshadow(ev) {  
  
    var mx;  
  
    var my;  
  
    if ( ev.layerX || ev.layerX == 0 ) {  
  
        mx= ev.layerX;  
  
        my = ev.layerY;  
  
    }  
  
    else if (ev.offsetX || ev.offsetX == 0) {  
  
        mx = ev.offsetX;  
  
        my = ev.offsetY;  
  
    }  
  
    can.style.cursor = "url(light.gif), pointer";  
  
    mx = mx+10;  
  
    my = my + 12;  
  
    drawshadowmask(mx,my);  
  
}
```

Now I needed to think what I wanted to do when the user pressed down on the mouse. I decided that I wanted the shadow to go away and the map to be

displayed in its full brightness. In addition to the appearance of things, I also wanted the Google Maps API to resume control. A critical reason for wanting the Google Maps API to take over is that I wanted to place a marker on the map, as opposed to the canvas, to mark a location. This is because I wanted the marker to move with the map, and that would be very difficult to do by drawing on the canvas. I would need to synchronize the marker on the canvas with panning and zooming of the map. Instead, the API does all this for me. In addition, I needed the Google Maps API to produce latitude and longitude values for the location.

The way to put Google Maps back in control, so to speak, was to “push the canvas under.” The function is

```
function pushcanvasunder(ev) {  
    can.style.zIndex = 1;  
  
    pl.style.zIndex = 100;  
}
```

The operation of pushing the canvas under or bringing it back on top is not instantaneous. I am open to suggestions on (1) how to define the interface and (2) how to implement what you have defined. There is room for improvement here.

One more situation to take care of is what I want to occur if and when the user

moves the mouse off from the canvas? The **mouseout** event is available as something to be listened for, so I wrote the code setting up the event (see the `can.addEventListener` statements shown above) to be handled by the **clearshadow** function. The clearshadow function accomplishes just that: clearing the whole canvas, including the shadow:

```
function clearshadow(ev) {  
    ctx.clearRect(0,0,600,400);  
}
```

In the function that brings in the Google Map, I set up an event handler for `mouseup` for maps.

```
listener = google.maps.event.addListener(map, 'mouseup', function(event) {  
    checkit(event.latLng);  
});
```

The call to **addListener**, a method that is part of the Google Maps API as opposed to JavaScript proper, sets up the call to the **checkit** function. The `checkit` function is invoked using an attribute of the `event` object as a parameter. As you can guess, `event.latLng` is the latitude and longitude values at the position of the mouse when the mouse button was released on the `map` object. The **checkit** function will use those values to calculate the distance from the base location and to print out the values along with the

distance on the screen. The code invokes a function I wrote that rounds the values. I did this to avoid displaying a value with many significant digits, more than is appropriate for this project. The Google Maps API **marker** method provides a way to use an image of my choosing for the marker, this time a black ,hand-drawn x, and to include a title with the marker. The title is recommended to make applications accessible for people using screen readers, though I cannot claim that this project would satisfy anyone in terms of accessibility. It is possible to produce the screen shown in Figure 4-19.

Figure 4-19. Title indicating distance shown on map

The **checkit** function, called with a parameter holding the latitude and longitude value, follows:

```
function checkit(clatlng) {  
  
    var distance = dist(clatlng,blatlng);  
  
    distance = round(distance,2);  
  
    var distanceString = String(distance)+" km";    marker = new  
    google.maps.Marker({  
  
        position: clatlng,  
  
        title: distanceString,  
  
        icon: bxmarker,
```

```
map: map });

var clat = clatlng.lat();

var clng = clatlng.lng();

clat = round(clat,4);

clng = round(clng,4);

document.getElementById("answer").innerHTML =

"The distance from base to most recent marker ("+clat+", "+clng+) is

"+String(distance)+" km.';

can.style.zIndex = 100; pl.style.zIndex = 1;

}
```

Notice that the last thing that the function does is put the canvas back on top of the map.

The CHANGE button and the radio buttons are implemented using standard HTML and JavaScript. The form is produced using the following HTML coding:

```
<form name="f" onSubmit=" return changebase();">

<input type="radio" name="loc" /> friends of ED, NYC<br/>

<input type="radio" name="loc" /> Purchase College<br/>

<input type="radio" name="loc" /> Illinois Institute of Technology<br/>
```

```
<input type="submit" value="CHANGE">  
</form>
```

The function **changebase** is invoked when the submit button, labeled CHANGE, is clicked. The **changebase** function determines which of the radio buttons was checked and uses the Locations table to pick up the latitude and longitude values. It then makes a call to **makemap** using these values for parameters. This way of organizing data is called *parallel structures*: the **locations** array elements correspond to the radio buttons. The last statement sets the **innerHTML** of the header element to display text, including the name of the selected base location.

```
function changebase() {  
  
    var mylat;  
  
    var mylong;  
  
    for(var i=0;i<locations.length;i++) {  
  
        if (document.f.loc[i].checked) {  
  
            mylat = locations[i][0];  
  
            mylong = locations[i][1];  
  
            makemap(mylat,mylong);  
  
            document.getElementById("header").innerHTML =
```

"Base location (small red x) is "+locations[i][2]; }

}

return false;

}

Calculating Distance and Rounding Values for Display

Google Maps, as many of us know, provides information on distances and even distinguishes between walking and driving. For this application, I needed more control on specifying the two locations for which I wanted the distance calculated, so I decided to develop a function in JavaScript. Determining the distance between two points, each representing latitude and longitude values, is done using the spherical law of cosines. My source was www.movable-type.co.uk/scripts/latlong.html. Here is

the code:

```
function dist(point1, point2) {  
  
    var R = 6371; // km  
  
    // var R = 3959; // miles  
  
    var lat1 = point1.lat()*Math.PI/180;  
  
    var lat2 = point2.lat()*Math.PI/180 ;  
  
    var lon1 = point1.lng()*Math.PI/180;  
  
    var lon2 = point2.lng()*Math.PI/180;  
  
    var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) +  
        Math.cos(lat1)*Math.cos(lat2) *
```

```
Math.cos(lon2-lon1)) * R;  
return d;  
}
```

Caution I don't include many comments in the code because I include the tables with each line annotated. However, comments are important. I strongly recommend leaving the comments on **km** and **miles** in the **dist** function so you can adjust your program as appropriate. Alternatively, you could display both values or give the user a choice.

The last function is for rounding values. When a quantity is dependent on a person moving a mouse, you shouldn't display a value to a great number of decimal places. However, we should keep in mind that latitude and longitude represent big units. I decided I wanted the distances to be shown with two decimal places and the latitude and longitude with four.

The function I wrote is quite general. It takes two parameters, one the number **num** and the other **places**, indicating how many decimal places to take the value. You can use it in other circumstances. It rounds up or down, as appropriate, by adding in the value I call the *increment* and then calculating the biggest integer not bigger than the value. So

- **round(9.147,2)** will produce 9.15; and
- **round(9.143, 2)** will produce 9.14.

The way the code works is first to determine what I term the *factor*, 10 raised to the desired number of places. For 2, this will be 100. I then calculate the increment. For two places, this will be $5 / 100 * 10$, which is $5 / 1,000$, which is .005. My code does the following:

1. Adds the increment to the original number
2. Multiplies the result by the factor
3. Calculates the largest whole number not bigger than the result (this is called the floor)—producing a whole number
4. Divides the result by the factor

The code follows:

```
function round (num,places) {  
    var factor = Math.pow(10,places);  
    var increment = 5/(factor*10);  
    return Math.floor((num+increment)*factor)/factor; }
```

I use the **round** function to round off distances to two decimal places and latitude and longitude to four decimal places.

Tip JavaScript has a method called **toFixed** that essentially performs the task of my round. If **num** holds a number—say, 51.5621—then **num.toFixed()** will produce 51 and **num.toFixed(2)** will produce 51.56. I've read that there can be inaccuracies with this method, so I chose to create my own function. You may be

happy to go with **toFixed()** in your own applications, though.

With the explanation of the relevant HTML5 and Google Maps API features,

we can now put it all together.

Building the Application and Making It Your Own

The map spotlight application sets up the combination of Google Maps functionality with HTML5 coding. A quick summary of the application is the following:

1. **init**: Initialization, including bringing in the map (**makemap**) and setting up mouse events with handlers: **showshadow**, **pushcanvasunder**, **clearshadow**
2. **makemap**: Brings in a map and sets up event handling, including the call to **checkit**
3. **showshadow**: Invokes **drawshadowmask**
4. **pushcanvasunder**: Enables events the on map
5. **checkit**: Calculates distance, adds a custom marker, and displays distance and rounded latitude and longitude

The function table describing the invoked/called by and calling relationships (Table 4-1) is the same for all the applications.

Table 4-1. Functions in the Map Maker Project

Function	Invoked/Called By	Calls

init	Invoked by action of the onLoad attribute in the body tag	makemap
pushcanvasunder	Invoked by action of addEventListener called in init	
clearshadow	Invoked by action of addEventListener called in init	
showshadow	Invoked by action of addEventListener called in init	drawshadowmask
drawshadowmask	Called by showshadow	
makemap	Called by init	
checkit	Called by action of addEventListener called in makemap	round, dist
round	Called by checkit (three times)	
dist	Called by checkit	
changebase	Called by action of onSubmit in <form>	makemap

Table 4-2 shows the code for the Map Maker application, named **mapspotlight.html**.

Table 4-2. Complete Code for the mapspotlight.html Application

Code Line	Description
<!DOCTYPE html>	Header
<html>	Opening html tag
<head>	Opening head tag
<title>Spotlight </title>	Complete title
<meta charset="UTF-8">	Meta tag
<style>	Opening of style element
header {font-family:Georgia,"Times New Roman",serif;	Set fonts for the heading
font-size:16px;	Font size

Code Line	Description
display:block; }	Line breaks before and after
	Style directive for the single

<code>canvas {position: absolute; top: 165px; left: 0px;</code>	canvas element: position slightly down the page
<code>z-index: 100;}</code>	Initial setting for canvas is on top of map
<code>#place {position: absolute; top: 165px; left: 0px;</code>	Style directive for the div holding the Google Map; position exactly the same as the canvas
<code>z-index: 1;}</code>	Initial setting to be under canvas
<code></style></code>	Close style element
<code><script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js?sensor=false"> </script></code>	Bring in the external script element holding the Google Maps API; no attempt to use sensing for geolocation
<code><script type="text/javascript" charset="UTF-8"></code>	Opening script tag

<code>var locations = [</code>	Define set of base locations
<code>[40.725592,-74.00495, "Friends of ED, NYC"],</code>	Latitude, longitude name for friends of ED
<code>[41.04796,-73.70539,"Purchase College/SUNY"],</code>	... Purchase College
<code>[41.878928, -87.641926,"Illinois Institute of Technology"]</code>	. . . Illinois Institute of Technology
<code>];</code>	Close array of locations
<code>var candiv;</code>	Used to hold div holding the canvas
<code>var can;</code>	Reference canvas element

Code Line	Description

<code>var ctx;</code>	Reference context of canvas; used for all drawing
<code>var pl;</code>	Reference the div holding the Google Map
<code>function init() {</code>	Function header for <code>init</code>
<code>var mylat;</code>	Will hold latitude value
<code>var mylong;</code>	Will hold longitude value
<code>candiv = document.createElement("div");</code>	Create a div
<code>candiv.innerHTML = ("<canvas id='canvas' width='600' height='400'>No canvas </canvas>");</code>	Set its contents to be a canvas element
<code>document.body.appendChild(candiv);</code>	Add to the body
<code>can = document.getElementById("canvas");</code>	Set reference to the canvas
<code>pl = document.getElementById("place");</code>	Set reference to the div holding the Google Map

<code>ctx = can.getContext("2d");</code>	Set the context
<code>can.onmousedown = function () { return false; } ;</code>	Prevents change of cursor to default
<code>can.addEventListener('mousemove',showshadow,true);</code>	Set event handling for mouse moving
<code>can.addEventListener('mousedown',pushcanvasunder,true);</code>	Set event handling for pushing down on mouse button
<code>can.addEventListener("mouseout",clearshadow,true);</code>	Set event handling for moving mouse off of the canvas
<code>mylat = locations[0][0];</code>	Set the latitude to be the latitude of the 0th location

Code Line	Description
<code>mylong = locations[0][1];</code>	Set the longitude to be the longitude of the 0th location
<code>makemap(mylat,mylong);</code>	Invoke function to make a map (bring in Google Maps at specified location)

<code>}</code>	Close init function
<code>function pushcanvasunder(ev) {</code>	Header for pushcanvas function, called with parameter referencing the event
<code>can.style.zIndex = 1;</code>	Push canvas down
<code>pl.style.zIndex = 100;</code>	Set map div up
<code>}</code>	Close pushcanvasunder function
<code>function clearshadow(ev) {</code>	Header for clearshadow function, called with parameter referencing the event
<code>ctx.clearRect(0,0,600,400);</code>	Clear canvas (erase shadow mask)
<code>}</code>	Close clearshadow function
<code>function showshadow(ev) {</code>	Header for showshadow function, called with parameter referencing the event
<code>var mx;</code>	Will be used to hold

	horizontal position of mouse
var my;	Will be used to hold vertical position of mouse
if (ev.layerX ev.layerX == 0) {	Does this browser use layerX ?

Code Line	Description
mx = ev.layerX;	If so, use it to set mx ...
my = ev.layerY;	... and my
} else if (ev.offsetX ev.offsetX == 0) {	Try offsetX
mx = ev.offsetX;	If so, use it to set mx ...
my = ev.offsetY;	... and my
}	Close clause
can.style.cursor = "url('light.gif'), pointer";	Set up cursor to be light.gif if available, otherwise pointer

mx = mx+10;	Make rough correction to make center of light at base of lightbulb horizontally and ...
my = my + 12;	... vertically
drawshadowmask(mx,my);	Invoke drawshadowmask function at the modified (mx,my)
}	Close showshadow function
var canvasAx = 0;	Constant for mask: Upper-left x
var canvasAy = 0;	Upper-left y
var canvasBx = 600;	Upper-right x
var canvasBy = 0;	Upper-right y
var canvasCx = 600;	Lower-right x
var canvasCy = 400;	Lower-right y

<code>var canvasDx = 0;</code>	Lower-left x
--------------------------------	--------------

Code Line	Description
<code>var canvasDy = 400;</code>	Lower-left y
<code>var holerad = 50;</code>	Constant radius for hole in shadow (radius of spotlight)
<code>var grayshadow = "rgba(250,250,250,.8)";</code>	Color for faint shadow; note alpha of .8
<code>function drawshadowmask(mx,my) {</code>	Header for drawshadowmask function; parameters hold center of donut hole
<code>ctx.clearRect(0,0,600,400);</code>	Erase whole canvas
<code>ctx.fillStyle = grayshadow;</code>	Set color
<code>ctx.beginPath();</code>	Start first (top) path
<code>ctx.moveTo(canvasAx,canvasAy);</code>	Move to upper-left corner
<code>ctx.lineTo(canvasBx,canvasBy);</code>	Draw over to upper-right corner

<code>ctx.lineTo(canvasBx,my);</code>	Draw to vertical point specified by my parameter
<code>ctx.lineTo(mx+holerad,my);</code>	Draw over to the left to edge of hole
<code>ctx.arc(mx,my,holerad,0,Math.PI,true);</code>	Draw semicircular arc
<code>ctx.lineTo(canvasAx,my);</code>	Draw to left side
<code>ctx.lineTo(canvasAx,canvasAy);</code>	Draw back to start
<code>ctx.closePath();</code>	Close path
<code>ctx.fill();</code>	Fill in
<code>ctx.beginPath();</code>	Start of second (lower) path
<code>ctx.moveTo(canvasAx,my);</code>	Start at point on left side indicated by my parameter

Code Line	Description
<code>ctx.lineTo(canvasDx,canvasDy);</code>	Draw to lower-left corner

<code>ctx.lineTo(canvasCx,canvasCy);</code>	Draw to lower-right corner
<code>ctx.lineTo(canvasBx,my);</code>	Draw to point on right edge
<code>ctx.lineTo(mx+holerad,my);</code>	Draw to left to edge of hole
<code>ctx.arc(mx,my,holerad,0,Math.PI,false);</code>	Draw semicircular arc
<code>ctx.lineTo(canvasAx,my);</code>	Draw to right edge
<code>ctx.closePath();</code>	Close path
<code>ctx.fill();</code>	Fill in
<code>}</code>	Close drawshadowmask function
<code>var listener;</code>	Variable set by addListener call; <i>not used again</i>
<code>var map;</code>	Holds map
<code>var blatlng;</code>	Holds base latitude longitude object
<code>var myOptions;</code>	Holds associative array used for map
<code>var rxmarker = "x1.png";</code>	Holds file name for red x image

<code>var bxmarker = "bx1.png";</code>	Holds file name for black x image
<code>function makemap(mlat,mylong) {</code>	Header for makemap function; parameters hold location of center of the map
<code>var marker;</code>	Will hold marker created for center

Code Line	Description
<code>blatlng = new google.maps.LatLng(mlat,mylong);</code>	Build a LatLng object (special data type for the API)
<code>myOptions = {</code>	Set associative array
<code>zoom: 12,</code>	Zoom setting (can be 0 to 18)
<code>center: blatlng,</code>	Center
<code>mapTypeId: google.maps.MapTypeId.ROADMAP</code>	Type of map

<code>};</code>	Close myOptions array
<code>map = new google.maps.Map(document.getElementById("place"), myOptions);</code>	Invoke the API to bring in a map at indicated place
<code>marker = new google.maps.Marker(</code>	Place marker in center of map; marker method takes an associative array as its parameter
<code>{</code>	Start of associative array
<code>position: blatlng,</code>	Set the position
<code>title: "center",</code>	Set the title
<code>icon: rxmarker,</code>	Set the icon
<code>map: map</code>	Set the map named parameter to the variable named map
<code>}</code>	Close the associative array which is the parameter to the call to Marker

<code>);</code>	Close the call to Marker
<code>listener = google.maps.event.addListener(</code>	Set up event handling (the three following parameters)
<code>map,</code>	The object, namely the map

Code Line	Description
<code>'mouseup',</code>	The specific event
<code>function(event) {</code>	An autonomous function (defined directly as a parameter in addListener)
<code>checkit(event.latLng);</code>	Calling checkit with the indicated latitude longitude object
<code>}</code>	Close the function definition
<code>);</code>	Close the call to addListener
<code>}</code>	Close the makemap function

function checkit(clatlng) {	Function header for checkit ; called with the latitude longitude object
var distance = dist(clatlng,blatlng);	Invoke dist function to calculate distance between the clicked position and the base
distance = round(distance,2);	Round the value
var distanceString = String(distance)+" km";	Set distanceString to be the display
marker = new google.maps.Marker(Invoke the Marker method, which takes an associative array as its parameter
{	Start of associative array
position: clatlng,	Set position
title: distanceString,	Set title
icon: bxmarker,	Set icon to be black x

Code Line	Description
map: map	Set map element of associative array to the value of the variable named map
}	Close associative array
);	Close call to Marker method
var clat = clatlng.lat();	Extract the latitude value
var clng = clatlng.lng();	Extract the longitude value
clat = round(clat,4);	Round value to 4 decimal places
clng = round(clng,4);	Round value to 4 decimal places
document.getElementById("answer").innerHTML =	Set up text on screen . . .
"The distance from base to most recent marker (" + clat+, "+clng+) is "+String(distance) +" km."	. . . to be calculated and formatted information
can.style.zIndex = 100;	Set canvas to be on top
pl.style.zIndex = 1;	Set pl (holding map) to be underneath

}	Close checkit function
function round (num,places) {	Header for function to round values
var factor = Math.pow(10,places);	Determine factor from number of places
var increment = 5/(factor*10);	Determine the increment to round up or down
return Math.floor((num+increment)*factor)/factor;	Do calculation
}	Close round function

Code Line	Description
function dist(point1, point2) {	Function header for dist (<u>distance</u>) function
// spherical law of cosines, // from // http://www.movable-type.co.uk/scripts/latlong.html	Attribution for my source. This is standard mathematics
var R = 6371; // km	Factor used to produce answer in kilometers
// var R = 3959; // miles	Commented out, but keep just in case you want to give

	answer in miles
<pre>var lat1 = point1.lat()*Math.PI/180;</pre>	Convert value to radians
<pre>var lat2 = point2.lat()*Math.PI/180 ;</pre>	Convert value to radians
<pre>var lon1 = point1.lng()*Math.PI/180;</pre>	Convert value to radians
<pre>var lon2 = point2.lng()*Math.PI/180;</pre>	Convert value to radians
<pre>var d =</pre>	Calculation ...
<pre>Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2- lon1)) * R;</pre>	Use trigonometry to determine distance
<pre>return d;</pre>	Return result
<pre>}</pre>	Close dist function
<pre>function changebase() {</pre>	Header for changebase function
<pre>var mylat;</pre>	Will hold new base location latitude

<code>var mylong;</code>	Will hold new base location longitude
<code>for(var i=0;i<locations.length;i++) {</code>	<code>for</code> loop to determine which radio button is checked

Code Line	Description
<code>if (document.f.loc[i].checked) {</code>	Is this one checked?
<code>mylat = locations[i][0];</code>	If so, set <code>mylat</code>
<code>mylong = locations[i][1];</code>	Set <code>mylong</code>
<code>makemap(mylat,mylong);</code>	Invoke <code>makemap</code>
<code>document.getElementById("header"). innerHTML = "Base location (small red x) is " + locations[i][2];</code>	Change text in header to show the name
<code>}</code>	Close <code>if true</code> clause
<code>}</code>	Close <code>for</code> loop

return false;	Return false to prevent refresh
}	Close function
</script>	Closing script tag
</head>	Closing head tag
<body onLoad="init();">	Opening body tag; include onLoad to invoke init
<header id="header">Base location (small red x) </header>	Semantic header element
<div id="place" style="width:600px; height:400px"></div>	Div to hold Google Maps
<div id="answer"></div>	Div to hold information on clicked locations
Change base location: 	Text
<form name="f" onSubmit=" return changebase();">	Start of form for changing

	base
<input type="radio" name="loc" /> Friends of ED, NYC 	Radio button choice
<input type="radio" name="loc" /> Purchase College 	Radio button choice

Code Line	Description
<input type="radio" name="loc" /> Illinois Institute of Technology 	Radio button choice
<input type="submit" value="CHANGE">	The button to make the change
</form>	Closing form tag
</body>	Closing body tag
</html>	Closing html tag

You need to decide on your set of base locations. Again, there is nothing special about three. If your base list is too large, you may consider using **<optgroup>** to produce a drop-down list. In any case, you need to define a set of locations. Each location has two numbers—latitude and longitude—and a string of text comprising the name. This text is repeated in the HTML coding in the form.

Testing and Uploading the Application

This project consists of the HTML file and three image files. For my version of the project, the image files were the lightbulb, **light.gif**; the red x, **rxmarker.png**; and the black x, **bxmarker.png**. There is nothing special about these image file types. You can use whatever you like. It could be argued that my x markers are too tiny, so think about your customers when deciding on what to do.

This application does require you to be online to test since that is the only way to make contact with Google Maps.

Summary

In this chapter, you learned how to do the following:

- Use the Google Maps API
- Combine Google Maps with canvas graphics
- Produce a GUI that includes Google Maps events and HTML5 events
- Draw using the alpha setting controlling transparency-opacity
- Change to a custom-made cursor
- Calculate distances between geographic points
- Round off decimal values for suitable display

The next chapter describes another project using Google Maps. You will learn how to build an application in which you can associate a picture, a video clip, or a picture-and-audio-clip combination with specific geographic locations, and display and play the specified media when a user clicks at or near the locations on a map.

CHAPTER 5

Map Portal: Using Google Maps to Access Your Media:

In this chapter, you will explore the following:

- Using the Google Maps API to play and display video, audio, and images
- Creating HTML5 markup dynamically
- Separating the program from descriptions of content
- Building a geography game

Introduction:

The projects in this chapter make use of the Google Maps API as a way to play video, display images, or play audio *and* display an image, all based on geographic locations. You can use these projects as models to build a study of a geographic area, report on a business or vacation trip, or create a geography quiz. I will describe three distinct applications. For all three applications, I have acquired media, such as video files, audio files, and image files, and I have defined in code an association between the files and specific geographic locations. To give you an idea of what I mean, for my projects, associations between a target location (which is given in latitude and longitude coordinates in the code) and media are shown in Table 5-1.

Table 5-1. Outline of Content

Description of Location	Media
Purchase College	Video from robotics class
Mount Kisco	Picture of Esther and an audio file of her playing piano
Dixon, Illinois	Picture of Aviva

Statue of Liberty, New York City

Video of fireworks

All three applications proceed smoothly with the different types of media. This is due to the features of HTML5 and, I say modestly, my programming. The media files are separate and the same for all three applications. It still is recommended that you supply multiple video and audio formats to make sure your application will work in the different browsers. The media types recognized by browsers may change so that fewer types are required, but this is not the case at this time.

The first application, consisting of one HTML file, named **mapvideos.html**, invites the viewer to click the map. If the click is close enough to one of the targeted locations, the viewer either sees the corresponding image, sees the corresponding video playing, or both sees the corresponding image and hears the corresponding audio file. The program handles all three types of media and media combinations.

The second application appears to the viewer to be the same as the first, but there is an important difference in the implementation. This application consists of two **html** files: **mapmediabase.html** and **mediacontent.js**. The **mediacontent.js** file holds the information describing the specific media content of the application. The **mapmediabase.html** program brings in the **mediacontent.js** file and creates during runtime what is necessary to replicate the first application.

I include both the first and second application because the first one is easier to understand, since all the information is in one place. When you develop your own applications, you can try to go straight to an implementation that separates coding from content, but don't feel too bad if you don't.

The third application is a quiz. Like the second, it consists of two files: **mapmediaquiz.html** and **mediaquizcontent.js**. The **mediaquizcontent.js** file contains information connecting the media to the locations and also contains the text for the questions.

The three applications have much of the same coding. The process of creating the second and third applications will show you how to scale up your applications by separating the details of a specific set of media (or other things) from the bulk of the programmin

Figure 5-1 shows the opening screen for the first application.

Click on map.

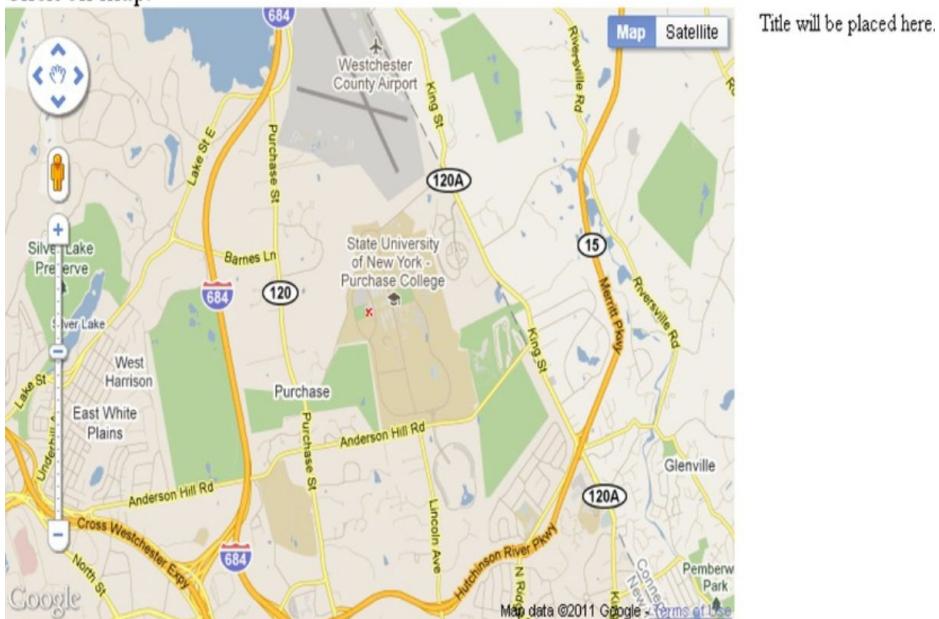
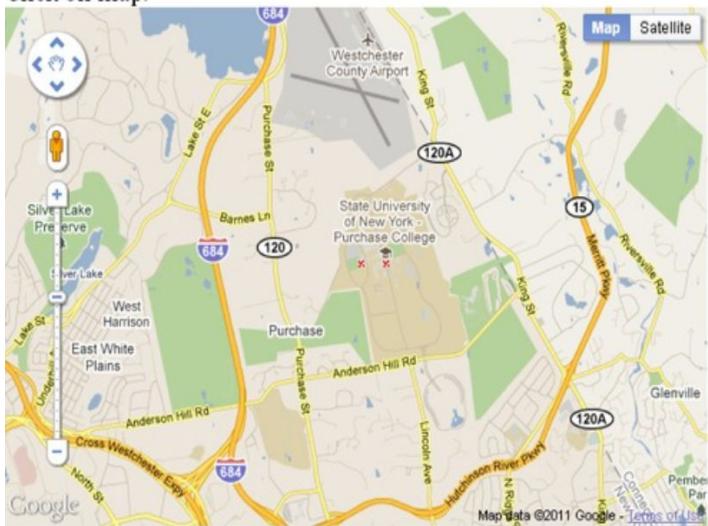


Figure 5-1. Opening screen

Figure 5-2 shows what happens when I click the screen on the Purchase College campus. A video clip appears and starts playing. You also can see “Lego robot” in the title position.

Click on map.

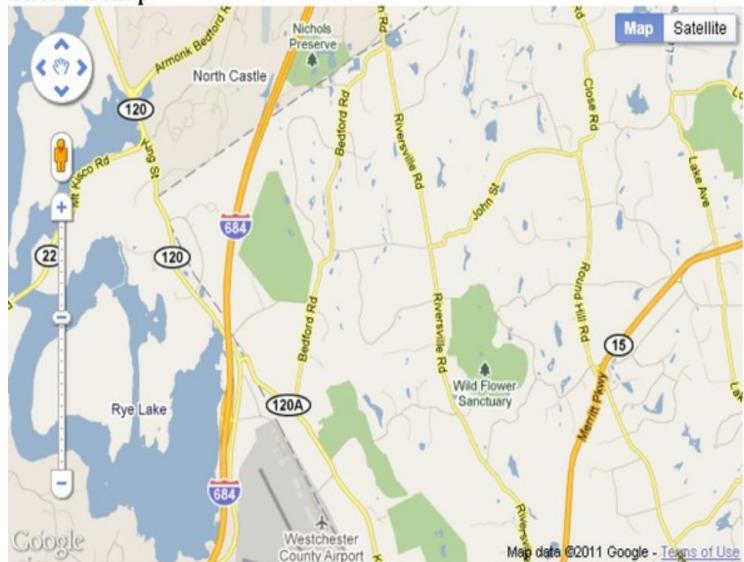


Lego robot



Figure 5-2. Result of clicking at Purchase College

Click on map.



Not close enough to any [new] target

Figure 5-3

shows the result of clicking the screen, but not sufficiently close to any of the target locations. Notice that I have panned the map, moving it to the north.

Figure 5-3. Click not close to any target

Figure 5-4 shows the result of my clicking near enough to Mt. Kisco to get a reward. I needed to move further north to get to Mt. Kisco. Notice also the audio control, providing a way to pause and resume playing and also change the speaker volume. The controls for audio (and video) will be different in the different browsers, but the functionality is the same.

Figure 5-4. Image-and-audio combination

Because I know where the locations are, I know to zoom out to get to the next location. Figure 5-5 shows the results of using the Google Maps interface to accomplish this. The audio track continues playing and I still see the picture.

Click on map.



Figure 5-5. Zooming out in preparation for a pan south

Figure 5-6 shows the result of moving the map to the south and then zooming in to the location of the Statue of Liberty, the targeted location for the fireworks video clip

Click on map.



Fire works



Figure 5-6. Clicking Liberty Island after panning and zooming in

Again, I know where the pictures are, so I zoom out, pan to the west of Chicago, and click the small town of Dixon, Illinois. Figure 5-7 shows the image I expected.

Click on map.



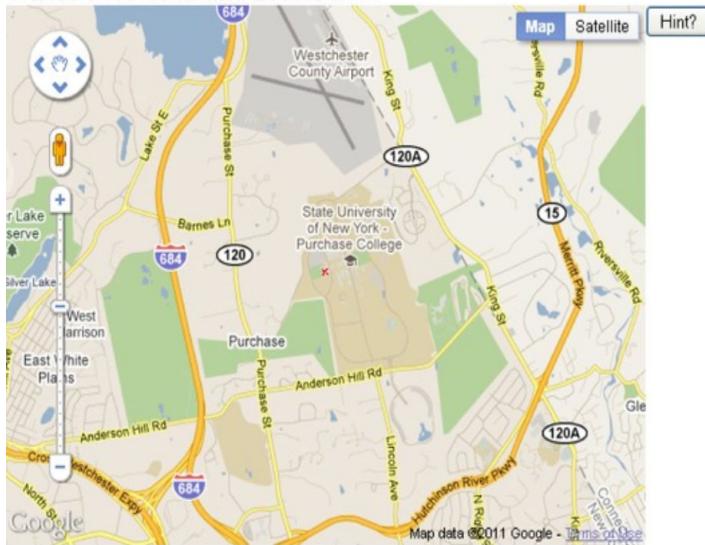
Aviva in Dixon



Figure 5-7. Panning to the west and zooming in to Dixon, Illinois

It actually took some work (to be explained later) to make the second application resemble the first with respect to layout.

Where does Grandma Esther live?



Title will be placed here.

Now I

will show screenshots for the third application, the quiz. Figure 5-8 shows the opening screen.

Figure 5-8. Opening screen of quiz

The question is, Where does Grandma Esther live? The player must click close to the location associated with this question. You might be able to guess the answer from the previous screenshots. Figure 5-9 shows an incorrect response. I just clicked the map near Purchase College.

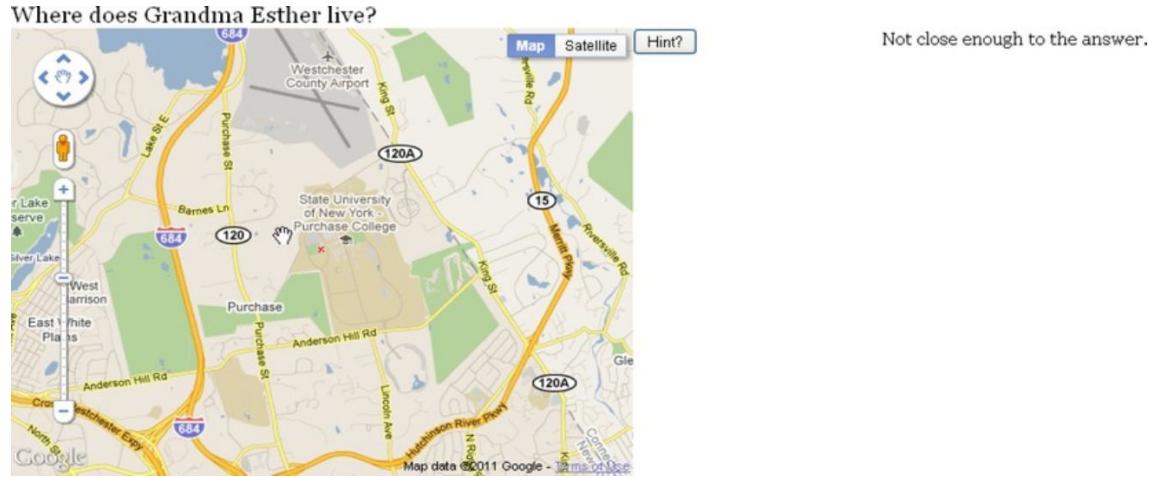


Figure 5-9. Incorrect response

I know the answer, and furthermore, how to get to it on the map. I move the map north to Mt. Kisco and click there. Figure 5-10 shows the familiar image

and the audio control. Piano music is playing.

Show the Lego robot navigating a maze.

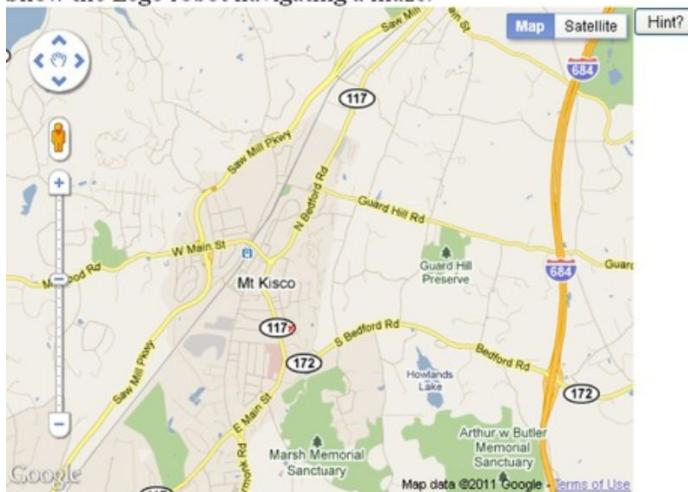


Figure 5-10. A correct answer is given, so the image is shown and the audio is played, and the next question is given.

Notice that the screen shows the next question as soon as the last one is answered correctly.

When designing a game such as this one, it is best to take pity on a player when they don't know the answer. I provide the Hint? button, though it goes beyond just giving a hint. Skipping ahead, I will get the next two questions

correct, and then I will need help on finding Dixon, Illinois. Figure 5-11 shows the prompt.

Figure 5-11. Prompt concerning flute playing

If I click the Hint? button, the application will bring in a new map, centered at the desired location. Figure 5-12 shows the screen.

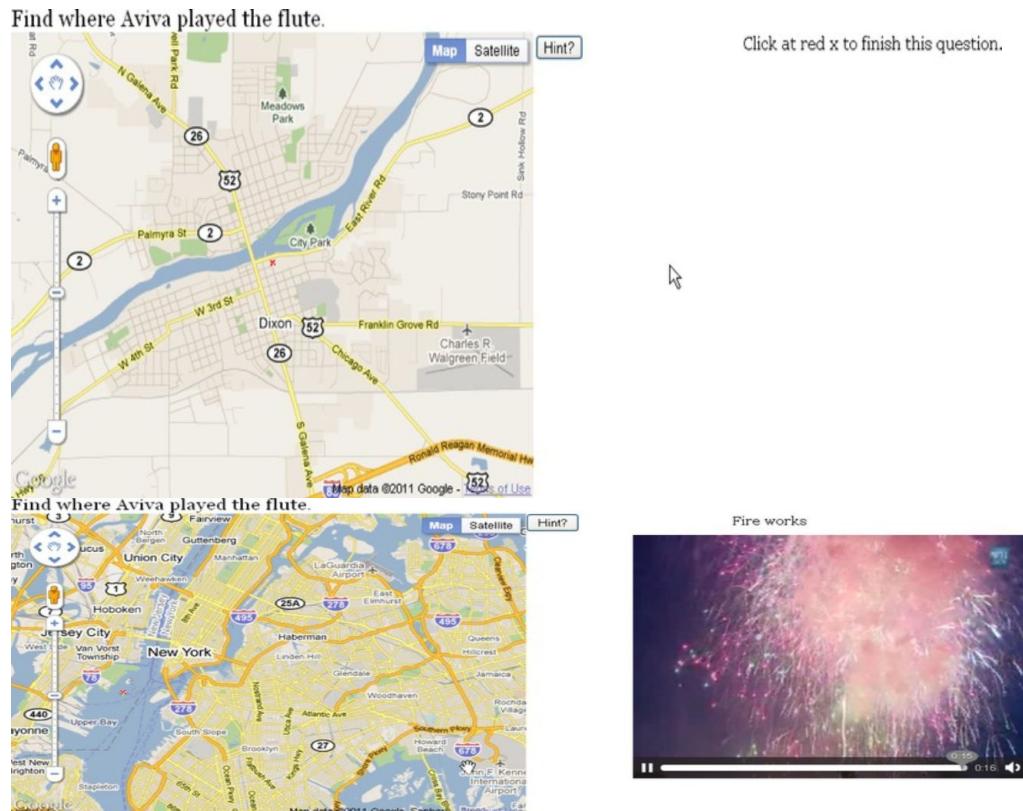


Figure 5-12. Map centered on Dixon, Illinois

It is still necessary to click the map, hopefully on or near the red x, to complete the question. You may say that there are better ways of hinting—such as supplying specifically chosen text holding the name of the place—and I won't argue with you. This is what I decided to do.

With this introduction, I'll go on to discuss the project history and the critical requirements.

Project History and Critical Requirements

A senior at Purchase College had collected and made video clips and photographs about the ethnic neighborhoods of Queens, New York, and wanted a way to present the work. The Google Maps API and the new facilities in HTML5 seemed perfect for the task. Keep in mind that the student only needed a way to present the work on a computer she set up at the senior project show, so the issue of noncompliant browsers was not a concern. The critical requirements include what is supplied by the Google Maps API. As you learned in the previous chapter, we can write code to access a map centered at a specified geographic location, set at an initial zoom level, and showing views of roads or satellite or terrain or a hybrid. In addition, the API provides a way to respond to the event of the viewer clicking the map. We need a way to define specific locations to be compared with the location corresponding to the viewer's click.

My first system for the student just used video and images. I later decided to add the image-and-audio combination. The critical requirement for the application is displaying and playing the designated media at the correct time *and* stopping and removing the media when appropriate, such as when it is time for the next presentation.

After developing the initial project, I thought of changes. The first one was the addition of the image-and-audio combination. I decided I did not want

audio just by itself. The next change was to separate the specific content from the general coding. This, in turn, required a way to create markup for video and audio elements dynamically.

I always like games and lessons, and it seemed like a natural step to build an application with questions or prompts that the viewer—now best described as player or student. The player gives the answer by finding the right position on the map. Both the general application and the quiz application have a requirement to define a tolerance with respect to the answers. The viewer/player/student cannot be expected to click exactly on the correct spot.

When testing the quiz, I realized I needed some way to help the player get past a particularly difficult question. Because I am a teacher, I decided to show the player the answer, rather than just skipping the question. However, as I indicated earlier, you may be able to devise a better way to produce hints.

Games should have some randomness feature, so I decided to shuffle the questions, though I did this somewhat later in the process.

Having described the critical requirements, the next section will contain explanation of the specific HTML5 features that can be used to build the projects.

HTML5, CSS, and JavaScript Features

Like the map maker project in Chapter 4, these projects are implemented by combining the use of the Google Maps API with features of HTML5. The combination for this project is not as tricky. The map stays on the left side of the window and the media is presented on the right. I will review quickly how to get access to a map and how to set up the event handling, and then go on to the HTML5, CSS, and JavaScript features for satisfying the rest of the critical requirements.

Google Maps API for Map Access and Event Handling

Access to the Google Maps API requires a script element with reference to an external file. For this application, I used

```
<script type="text/javascript" charset="UTF-8"
src="http://maps.google.com/maps/api/js?
sensor=false"></script>
```

I set up the map using a function I named **makemap**. It has two parameters: two decimal numbers that represent the latitude and longitude values:

```
function makemap(mylat, mylong)
```

The global variables **zoomlevel**, holding a number from 0 to 18, and **xmarker**, holding the address of an image file, are set before the function **makemap** is invoked.

The code to bring in a map is an invocation of the **google.maps.Map** constructor method. It takes two parameters. The first is the location in the HTML document where the map is to appear. I set up a div with ID **place** in the body of the document:

```
<div id="place" style="float: left; width: 50%; height: 400px"></div>
```

The second parameter is an associative array. The following three statements set up the location at which the map is centered as a Google Maps

latitude/longitude object, create the associative array **myOptions**, and invoke the **Map** constructor:

```
blatlng = new google.maps.LatLng(mylat,mylong);
```

```
myOptions = {
```

```
zoom: zoomlevel,
```

```
center: blatlng,
```

```
mapTypeId: google.maps.MapTypeId.ROADMAP
```

```
};
```

```
map = new google.maps.Map(document.getElementById("place"), myOptions);
```

For completeness sake, here are screenshots with other settings for the map type. These are TERRAIN, HYBRID, and SATELLITE. Figure 5-13 shows the results of requesting the setting showing the terrain—that is, colors indicating elevations, water, park, and human-constructed areas:

mapTypeId: google.maps.MapTypeId.TERRAIN

Figure 5-13. TERRAIN map type

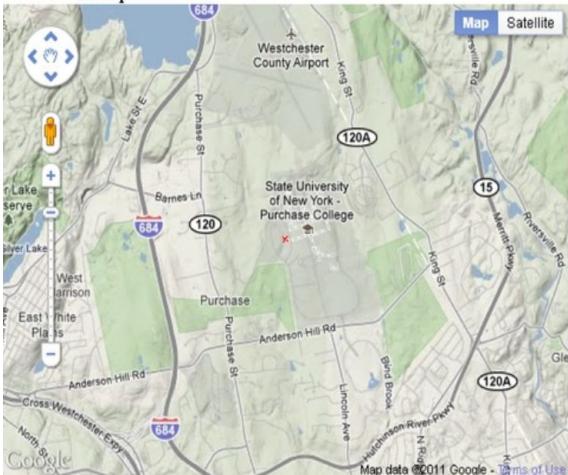
Figure 5-14 shows the results of requesting the HYBRID view, combining satellite and road map imagery.

mapTypeId: google.maps.MapTypeId.HYBRID

Figure 5-14. HYBRID map type

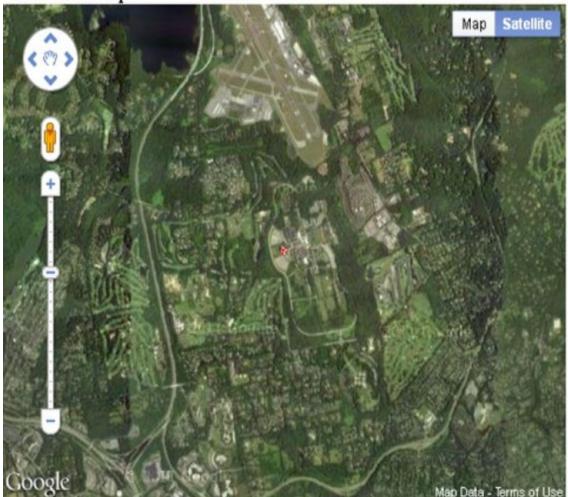
By the way, the hybrid map is what is produced by clicking the Satellite option on the interface. Figure 5-15 shows the results of requesting SATELLITE images.

Click on map.



Title will be placed here.

Click on map.



Title will be placed here.

mapTypeId:

google.maps.MapTypeId.SATELLITE

Figure 5-15. SATELLITE map type

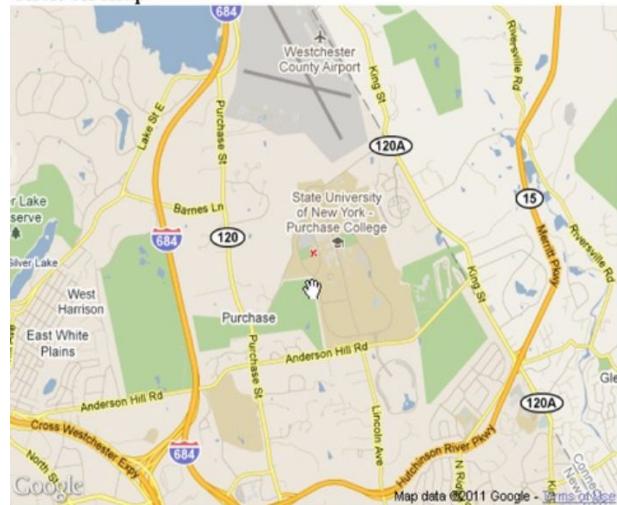
Lastly, you may have an application in which you do not want the viewer to change the map. You can prevent the user from changing the map by disabling the default interface with the use of an additional option in the **myOptions** array. Note the comma after ROADMAP.

```
mapTypeId: google.maps.MapTypeId.ROADMAP,
```

```
disableDefaultUI: true
```

Figure 5-16 shows the results.

Click on map.



Title will be placed here.

Figure 5-16. Map interface removed

There are two more operations for **makemap** to carry out. A custom marker is placed on the map at the indicated center location and event handling is set up for clicking the map:

```
marker = new google.maps.Marker({ position: blatlng, title:  
"center",  
  
icon: xmarker, map: map }));  
  
listener = google.maps.event.addListener(map, 'click',  
    function(event) {  
        checkit(event.latLng);  
    });
```

The **xmarker** value references an Image object that has its **src** set to an external file named **x1.png**.

Project Content

The portal projects all present media connected to a map location. My projects use three types of media: **video**, **picture**, and **pictureaudio**. Note: these are my terms for the three types I have chosen to include in the project. The content of the portal projects is specified using an array I named **content**. Each element of the array is itself an array five or six elements. The first four elements are the same for all the types: the latitude, longitude, title, and type. The fifth or the fifth and the sixth point to the specific media elements. The array

```
content = [  
    [41.19991,-73.72353,"Esther      at      home","pictureaudio",esther,aud1],  
    [41.05079,-73.70448,"Lego robot ","video",vid1],  
    [40.68992,-74.04460,"Fire works ","video",vid2], [41.8442,-89.480,"Aviva in  
Dixon","picture",aviva] ];
```

specifies four locations, starting with a picture/audio combination, followed by two videos, followed by one picture. The element in the array for the picture/audio combination includes, as you would expect, two additional pieces of information. It is not obvious from just this section of code, but **esther** refers to an Image element and **aud1** refers to an audio element. Similarly, **vid1** and **vid2** refer to video elements, and **aviva** refers to another

Image element.

The **content** array is referenced by the **checkit** function (to be described following), and the appropriate media are presented.

The quiz project has additional content, namely an array of questions, to be shown later.

Presentation and Removal of Video, Audio and Images

I assume you have a basic understanding of the HTML5 features for presenting video and audio and for drawing images from files on the canvas. The map portal projects require code that presents the media on demand (i.e., making the media appear and go away). This requirement is similar to the bouncing video of Chapter 3. In the basic **mapvideos** application, I included the definitions of the video, audio, and canvas elements in the body element of the document, all within a div element that I specified as floating to the right:

```
<div style="float: right; width: 38%; height: 400px">

<div id="answer">Title will be placed here.</div>

<p> </p>

<video id="maze" preload="auto" controls="controls" width="400">

<source src="maze.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>

<source src="maze.theora.ogv" type='video/ogg; codecs="theora, vorbis"'> <source
src="maze.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'>

Your browser does not accept the video tag.

</video>

<video id="fire" preload="auto" controls="controls">

<source src="sfire3.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2'">
```

```

<source    src="sfire3.theora.ogv"    type='video/ogg;    codecs="theora,    vorbis"'>    <source
src="sfire3.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'>

Your browser does not accept the video tag.

</video>

<audio id="mpiano" controls="controls" preload="preload">

<source src="estherT.ogg" type="audio/ogg" />

<source src="estherT.mp3" type="audio/mpeg" />

<source src="estherT.wav" type="audio/wav" /> </audio>

<canvas id="canvas" width="300" height="300" > Your browser doesn't recognize canvas </canvas>

</div>

```

Notice that I did not use the **loop** attribute in the **<video>** tag, nor did I put in a call to **addEventListener** to a function to restart the video, based on the assumption that **loop** is not recognized by some browsers. This is because the video controls are visible, and if the viewer wants to replay the video, he or she can do it. Similarly, the viewer can re-play the audio clip.

In the style element, I put directives to make all video not display, and for the positioning to be relative:

```

video
{display:none;
position:relative;
}
audio {display:none; position:relative;}

```

I did not have to do something analogous for the canvas. The canvas is always present, but if nothing has been drawn on it or it has been cleared, then nothing is visible.

The critical coding for presenting the media is in a **switch** statement within the **checkit** function. The **bestyet** variable holds the index of the element in the **content** array that is the closest to the location where the viewer/player clicked the map. If it has been determined to be close enough, then it will proceed with displaying the picture and playing the video or audio.

```
switch (content[bestyet][3])  
{ case "video":  
    answer.innerHTML=content[bestyet][2];  
    v = content[bestyet][4];  
    v.style.display="block";  
    v.currentTime = 0;  
    v.play();  
  
    break;  
    case "picture":  
    case "pictureaudio":
```

```
answer.innerHTML=content[bestyet][2];  
  
ctx.drawImage(content[bestyet][4],10,10);  
  
if (content[bestyet][3]=="picture") {  
  
break;}  
  
else {  
  
audioel = content[bestyet][5];  
  
audioel.style.display="block";  
  
audioel.currentTime = 0;  
  
audioel.play();  
  
break;  
  
}  
  
}
```

Changing the display style to "**block**" has the effect of displaying the video or audio controls. Assigning 0 to the **currentTime** means that the video or audio will play from the start. You can make use of **currentTime** to produce other effects—for example, playing different parts of a long audio or video clip.

There is one more piece of coding that is required. This is removing the last-viewed content: stopping the video or audio, and stopping the display. The code to do this must take account of the chance that there is nothing to

remove. The code is'

```
if (v != undefined) {  
    v.pause();  
    v.style.display = "none"; }  
  
if (audioel != undefined) {  
    audioel.pause();  
    audioel.style.display = "none";  
}  
  
ctx.clearRect(0,0,300,300);
```

Note that there is no harm in clearing the canvas even if there was nothing drawn on it. There would be an error in the statement **v.pause();** if **v** was not set. For the quiz project, I put this code into its own function, which I named **eraseold**, because removing the old material needs to be done at two different places in the code.

Distances and Tolerances

The calculation of distance between two latitude/longitude points was described in the previous chapter. The issue to be explained here concerns how to make comparisons of distances. For the portal application, I need to write code to determine the location specified in the **content** array that is closest to the position the viewer clicked. I do this using a **for** loop. As is typical in these “best so far” calculations, I start off the process by computing the distance to the 0th (i.e., the first) element in the **content** array. This is the best so far. The **for** loop then proceeds, starting with index 1.

```
function checkit(clatlng) {  
  
    var i;  
  
    var latlnga = new google.maps.LatLng(content[0][0],content[0][1]);  
  
    var bestyet = 0;  
  
    var closestdistance = dist(clatlng,latlnga);  
  
    var distance;  
  
    for (i=1;i<content.length;i++) {  
  
        latlnga = new google.maps.LatLng(content[i][0],content[i][1]);  
  
        distance = dist(clatlng,latlnga);  
  
        if (distance < closestdistance) {  
  
            closestdistance = distance;  
  
            bestyet = i;  
  
        }  
    }  
}
```

```
}
```

When the **for** loop is complete, **bestyet** holds the index to the best yet, meaning the closest, and **closestdistance** holds the distance to that element, which has been determined to be the closest (smallest) distance.

I then need to write code that checks if that element—the one with index named in **bestyet**—is close enough to proceed. In gaming and other applications, the term *tolerance* or *margin* are used. You can't expect a person to click exactly on a location. That is not possible when the units are pixels, and it definitely isn't possible when the units are latitude and longitude and the exact point may not be available to the user under the current zoom level. The variable **maxdistance** holds the value that I choose to use for this test. Here is the rest of the **checkit** function (without repeating the whole **switch** statement):

```
if (distance < maxdistance) {  
  
marker = new google.maps.Marker({  
  
position: clatlng,  
  
title: content[bestyet][2],  
  
icon: xmarker,  
  
map: map});  
  
switch (content[bestyet][3]) {  
  
...  
}
```

```
}

else {

answer.innerHTML="Not close enough to any [new] target";

}

}
```

Regular Expressions

Regular expressions are a powerful facility for describing patterns of character strings (text) for checking and for manipulation. It is a whole language for specifying patterns. For example, to give you a flavor of this large topic, the pattern

```
/^5[1-5]\d{2}-?\d{4}-?\d{4}-?\d{4}$/
```

can be used detect MasterCard numbers. These numbers start with 51 to 55, followed by two more digits, and then three groups of four digits. This pattern accepts the dashes, but does not require them. The `^` symbol means the pattern must be present at the start of the string, and the `$` means it must go to the end of the string. The forward slashes (`/`) are delimiters for the pattern and the backslashes are escape characters. Interpreting this pattern starting at the start goes as follows:

- `^` : Start at the start of the string.
- `5` : Pattern must contain a 5.
- `[1-5]`: Pattern must contain one of the numbers 1, 2, 3, 4, or 5.
- `\d{2}`: Pattern must contain exactly two digits.
- `-?`: Pattern must contain 0 or 1 -.
- `\d{4}`: Pattern must contain exactly four digits.

- -?: Pattern must contain 0 or 1 -.
- \d{4}: Pattern must contain exactly four digits.
- -?: Pattern must contain 0 or 1 -.
- \d{4}: Pattern must contain exactly four digits.
- \$: End of string.

MasterCard numbers must obey other rules as well, and you can do the research to find out how to verify them further. Don't worry, we'll be using a much simpler regular expression (also known as a *regex*) than that.

The use of regular expressions predates HTML. Regular expressions can be used in forms to specify the format of the input. For this application, we will use the **replace** method for strings to find all instances of a specific small piece of text within a long string and replace it with something else. One of the statements I use is

```
videomarkup = videomarkup.replace(/XXXX/g,name);
```

What this does is find all occurrences (this is what the **g** does) of the string **XXXX** and replace them with the value of the variable **name**.

I could and probably should have made even more use of regular expressions to verify the data defining the content of the applications. Maybe that's something you want to experiment with in your own applications.

External Script File

For this project and the one in the previous chapter, I demonstrated the use of a script element to bring in an external script file, namely the Google Maps API. You can also use this facility to bring in your own external file. My goal is to put all the specific content relating to my project in its own file. For the mapmediabase case, this is a file I named mediacontent.js. It follows in its entirety:

```
var base= [41.04796,-73.70539,"Purchase College/SUNY"];  
  
var zoomlevel = 13;  
  
var precontent = [  
  
[41.19991,-73.72353,"Esther at home","pictureaudio","estherT","esther.jpg"],  
[41.05079,-73.70448,"Lego robot ","video","maze"],  
[40.68992,-74.04460,"Fire works ","video","sfire3"],  
[41.8442,-89.480,"Aviva in Dixon","picture","avivadixon.jpg"]  
];  
  
var maxdistance = 5;
```

Tip The next step could be putting the information on the media content in a database.

The **base**, **zoomlevel**, and **maxdistance** variables are all what they seem. The base is the initial center point for the map. The zoom level specifies the initial zoom. I say *initial* because the user can use the Google Maps controls to pan

or zoom in or out. The **maxdistance** is the number I use to check if the user clicks close enough to one of the locations. You will need to determine what the appropriate distance is for your application.

I simply moved the variable declarations from the other document into what will be the external, content-specific document. I must admit that I forgot about **zoomlevel** and **maxdistance**, and only moved them after thinking about what other projects might be. For example, you may decide to build a project with a very different zoom level. Maybe you want your player to distinguish at the city block level, in which case the maximum distance might be 1 kilometer or less.

The **precontent** array resembles but is not the same as the **content** array. What is different is that the fields that now have "**estherT**", "**esther.jpg**", "**maze**", "**sfire3**", and "**avivadixon.jpg**" are used to create HTML5 elements or JavaScript elements. Once these are created, then references can be inserted in a **content** array. For this to work, I needed to change the names of the video and the audio files so that the video element coding and the audio element coding all fit a pattern.

For the mapmediaquiz application, I needed to add the questions. I could have added another element to the inner arrays of the **precontent** array, but decided against since that would require changes in the coding. Instead, I defined another array called **questions**. This is called *parallel structures* and is a

common technique.

Note At some point, the right decision may be to stop using straight JavaScript arrays, including the use of parallel structures, and make use of XML or a database. I didn't think it was called for in this application, but I could be wrong. Note that the use of databases does provide a way to hide the data

```
Var base= [41.04796,-73.70539,"Purchase College/SUNY"];  
  
var zoomlevel = 13;  
  
var precontent = [  
  
[41.19991,-73.72353,"Esther at home","pictureaudio","estherT","esther.jpg"],  
  
[41.05079,-73.70448,"Lego robot ","video","maze"],  
  
[40.68992,-74.04460,"Fire works ","video","sfire3"],  
  
[41.8442,-89.480,"Aviva in Dixon","picture","avivadixon.jpg"]  
];  
  
var questions = [  
  
"Where does Grandma Esther live?",  
  
"Show the Lego robot navigating a maze.", "Where are great fireworks?",  
  
"Find where Aviva played the flute."  
];  
  
var maxdistance = 10;
```

The external scripts are brought into the main document using a script element. For the mapmediabase program, this is

```
<script type="text/javascript" src="mediacontent.js"> </script>
```

and for mapmediaquiz, this is

Now I will explain how to use the precontent information to build what is required to make the projects work.

Dynamic Creation of HTML5 Markup and Positioning

The external script statements bring in the information for the base and the quiz applications. Now is the time to explain how the information is used. In both cases, the **init** function will invoke a function I named **loadcontent**. This function calls **makemap** to make a map at the indicated base location.

```
makemap(base[0],base[1]);
```

The **content** array starts off as an empty array.

```
var content = [];
```

By the way, this is different from

```
var content;
```

Your code needs to make **content** an array.

It then uses a **for** loop to iterate over all the elements of **precontent**. The start of the **for** loop adds the *i*th element of **precontent** to the **content** array.

```
for (var i=0;i<precontent.length;i++) {  
    content.push(precontent[i]);  
    name = precontent[i][4];
```

The next line is the header of a **switch** statement using as the condition the element of the inner arrays that indicates the type

```
switch (precontent[i][3]) {
```

For video and pictureaudio, the code creates a div element and positions it so that it floats to the right. It then places inside the div element the right markup for video or audio. What is that markup? I have what I will describe as dummy strings that have XXXX where the actual names of the video or audio files would go. These strings are

```
var videotext1 = "<video id=\"XXXX\" preload=\"auto\"  
controls=\"controls\" width=\"400\"><source src=\"XXXX.mp4\"  
type='video/mp4';  
  
codecs=\"avc1.42E01E,mp4a.40.2\"';  
var videotext2=<source src=\"XXXX.theora.ogv\" type='video/ogg';  
  
codecs=\"theora,vorbis\"';  
src=\"XXXX.webmvp8.webm\" type='video/webm;  
  
codec='vp8, vorbis\"';  
  
var videotext3="Your browser does not accept the video tag.</video>";  
  
var audiotext1=<audio id=\"XXXX\" controls=\"controls\"  
preload=\"preload\"><source src=\"XXXX.ogg\" type='audio/ogg' />";  
  
var audiotext2=<source src=\"XXXX.mp3\" type='audio/mpeg' /><source  
src=\"XXXX.wav\" type='audio/wav' /></audio>";
```

I divided the strings into sets of three and two just to make it easier for me to check. Notice the use of the backslash (\) It tells JavaScript to use the next symbol as is, and not interpret it as a special operator for regular expressions. This is how the quotation marks inside the screen get carried over to be part of the HTML.

My approach required that I make sure that the names of the video and audio files follow this pattern. This meant that the mp4 files all needed to contain just the name and no internal dots.

I write code using the regular expression function replace to take information out of the **precontent** array and put it in the strings in as many places as necessary. The **switch** statement in its entirety is

```
switch (precontent[i][3]) {  
  
    case "video":  
  
        divelement= document.createElement("div");  
  
        divelement.style = "float: right; width:30%;";  
  
        videomarkup = videotext1+videotext2+videotext3;  
  
        videomarkup = videomarkup.replace(/XXXX/g,name);  
  
        divelement.innerHTML = videomarkup;  
  
        document.body.appendChild(divelement);  
  
        videoelementreference = document.getElementById(name);  
  
        content[i][4] = videoelementreference;
```

```
break;

case "pictureaudio":
    divelement = document.createElement("div");
    divelement.style = "float: right; width: 30%;";
    audiomarkup = audiotext1 + audiotext2;
    audiomarkup = audiomarkup.replace(/XXXX/g, name);
    divelement.innerHTML = audiomarkup;
    document.body.appendChild(divelement);
    audioreference = document.getElementById(name);
    savedimagefilename = content[i][5];
    content[i][5] = audioreference;
    imageobj = new Image();
    imageobj.src = savedimagefilename;
    content[i][4] = imageobj;
    break;

case "picture":
    imageobj = new Image();
    imageobj.src = precontent[i][4];
    content[i][4] = imageobj;
    break;
}
```

Notice that the **pictureaudio** case does some juggling to create the content element with references to the newly created audio element and the Image element.

However, this was not quite enough to ensure that the video and audio end up on the right-hand side for all browsers. That is, it worked for some but not others. I decided to position the audio and video exactly—that is, in absolute terms. This required the following CSS in the style element for all video and audio elements:

```
video {display:none; position:absolute; top: 60px; right: 20px; }
```

```
audio {display:none; position:absolute; top: 60px; right: 20px;}
```

The position of the audio is for the audio controls.

Hint Button

You can tell from my coding that I was ambivalent about whether to provide a hint or help a player who had given up. In the body element, I included

```
<button onClick="giveup();">Hint? </button>
```

The **giveup** function creates a new map. That is, it uses the **makemap** function to construct access to a different Google Map in the same place. It also erases the old media and puts directions into the answer element.

```
function giveup() {  
  makemap(content[nextquestion][0],content[nextquestion][1]);  
  eraseold();  
  answer.innerHTML="Click at red x to finish this question.";  
}
```

Shuffling

I added a shuffle step to mix up the order of questions. More exactly, the program shuffles the **content** array and the **questions** array, keeping the items in parallel. The shuffle algorithm I used is the same Fisher-Yates algorithm demonstrated in Chapter 10 of *The Essential Guide to HTML5*. See also <http://eli.thegreenplace.net/2010/05/28/the-intuition-behind-fisher-yates-shuffling/> for an explanation on how the shuffling works.

This algorithm determines a random location for each element. Since I need to keep the two arrays in parallel, the same swap operation must be made for each array. In the code that follows, you see that it is determined that the **s** and **i** elements are to change locations. Two variables, **hold** and **holdq**, are the extra places used in the swap operation.

```
function shufflecontent() {  
    var i=content.length-1;  
  
    var s;  
  
    var hold;  
  
    var holdq; while(i>0) {  
  
        s = Math.floor(Math.random()*(i+1));  
  
        hold = content[s]; content[s]=content[i];  
  
        content[i] = hold; holdq = questions[s];  
    }  
}
```

```
questions[s]=questions[i];  
  
questions[i] = holdq;  
  
i--;  
  
}  
  
}
```

With this explanation of the various parts, I'll go on in the next section to describe the three applications.

Building the Application and Making It Your Own

The first and critical step in making the application your own is to decide on the content. You can choose to not implement the first application, mapvideos, which has the content hard-coded, so to speak, with all the other coding. Again, I included it because I wrote it first and it is the easiest to understand. I strongly believe that most people would develop something like it, and then possibly decide to separate content from coding. You may go straight to mapmediabase or mapmediaquiz if you like.

The Mapvideos Application

The **mapvideos** application sets up the combination of Google Maps functionality with HTML5 coding for video, audio, and pictures on canvas. The canvas is not on top of the map. A quick summary of the application follows:

1. **init**: Performs initialization, including invoking **makemap** for bringing in the map. The **init** function constructs the **content** array using data constructed as references to elements in the body.
2. **makemap**: Brings in a map and sets up event handling, including the call to **checkit**.
3. **checkit**: Compares the clicked location with the locations described in the **content** array. If one is close enough, then the associated media is shown and played.
4. **dist**: Computes the distance between two locations.

Table 5-2 outlines the functions in the mapvideos project. The function table describing the invoked/called by and calling relationships is similar for all the applications.

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the body tag	

		makemap
makemap	Invoked by init	
checkit	Invoked by addListener call in makemap	
dist	Invoked by checkit	dist

Table 5-2. Functions in the Mapvideos Portal Project

Table 5-3 shows the code for the original portal application, **mapvideos.html**.

Table 5-3. Complete Code for the Mapvideos Portal Application

Code Line	Description
<!DOCTYPE html>	Doctype for HTML5
<html>	html tag
<head>	Head tag
<title>Clickable map </title>	Complete title element
<meta charset="UTF-8">	Meta tag, standard for HTML5

<style>	Style tag
header {font-family:Georgia, "Times New Roman",serif;	Set styling for the header, a semantic element; the font family makes Georgia the first choice, with Times New Roman a fallback, and the default serif the next fallback choice of fonts
font-size:20px;	Fairly big font
display:block;	Set line breaks before and afterward

}	Close style directive
video {display:none; position:relative; }	Style directive for video
audio {display:none; position:relative;}	Style directive for audio; note that this is for the controls
</style>	Closing style tag

<pre><script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js? sensor=false"></script></pre>	Script element bringing in Google Maps API
<pre><script type="text/javascript" charset="UTF-8"></pre>	Starting script tag
<pre>var maxdistance = 5;</pre>	Set maxdistance to be 5, a value I decided was appropriate; a click needed to be within 5 kilometers of one of the targets to be considered close enough
<pre>var listener;</pre>	Placeholder for call of addListener
<pre>var map;</pre>	Variable holding the current map
<pre>var myOptions;</pre>	Variable for the associative array holding the options for a call to the Map constructor
<pre>var ctx;</pre>	Variable holding the context of the canvas
<pre>var blatlng;</pre>	Variable holding the constructed

	latitude/longitude object for the base location
<code>var esther = new Image();</code>	Variable holding an Image object
<code>esther.src = "esther.jpg";</code>	Set the src of this Image object to the file address
<code>var aviva = new Image();</code>	Variable holding an Image object
<code>aviva.src = "avivadixon.jpg";</code>	Set the src of this Image object to the file address
<code>var vid1;</code>	Set in init to one of the video elements
<code>var vid2;</code>	Set in init to one of the video elements

<code>var aud1;</code>	Set in init to the audio element
<code>var content;</code>	Declared here as global variable; will be set in init
<code>var answer;</code>	Will be set in init to reference a div in the

	body
var v;	reference to current (last) video
var audioel;	Reference to current (last) audio
var base= [41.04796,-73.70539, "Purchase College/SUNY"];	Variable set to the base for my project
function init() {	Function header for init
ctx = document.getElementById("canvas").getContext('2d');	Set ctx for use in drawing on canvas
');	
makemap(base[0],base[1]);	Invoke makemap
answer = document.getElementById("answer");	Set answer
vid1 = document.getElementById("maze");	Set vid1 for one of the video elements
vid2 = document.getElementById("fire");	Set vid2 for the other video element
aud1 = document.getElementById("mpiano");	Set aud1 for the audio element
content = [Set the setting of the content array
[41.19991,-73.72353,"Esther at home","pictureaudio",esther,aud1],	Info for the Esther picture audio combination
[41.05079,-73.70448,"Lego robot<br"></br">, "video",vid1],	Info for the Lego robot video
[40.68992,-74.04460,"Fire works<br"></br">, "video",vid2],	Info for the Fireworks video

[41.8442,-89.480,"Aviva in Dixon","picture",aviva]	Info for the Aviva picture
];	End of content array

}	Close init function
var xmarker = "x1.png";	Set file address for marker on the map
function makemap(mlat,mylong) {	Header for makemap function; map to be centered (mlat,mylong)
var marker;	Holds the constructed marker
blatlng = new google.maps.LatLng(mlat,mylong);	Set the latitude/longitude object
myOptions = {	Start to set up the options array
zoom: 13,	Zoom set at constant, arrived at after experimenting
center: blatlng,	Center at blatlng
mapTypeId: google.maps.MapTypeId.ROADMAP	Roadmap type

<code>};</code>	Close myOptions array
<code>map = new google.maps.Map(document.getElementById("place"), myOptions);</code>	Invoke constructor to bring in the Google Maps
<code>marker = new google.maps.Marker({ position: blatlng, title: "center", icon: xmarker, map: map });</code>	Mark center
<code>listener = google.maps.event.addListener(map, 'click', function(event) {</code>	Set up event handling for clicking the map
<code> checkit(event.latLng);</code>	The function checkit is invoked with the latLng attribute of the event as the parameter
<code>});</code>	Close call to bring in map
<code>}</code>	Close makemap function
<code>function checkit(clatlng) {</code>	Header for checkit function
<code>var i;</code>	Used for indexing

<pre>var latlnga =new google.maps.LatLng(content[0][0],content[0][1]);</pre>	Build latitude/longitude object from first element of content
<pre>var bestyet = 0;</pre>	Will point to the index determined to be the best (closest) so far
<pre>var closestdistance = dist(clatlng,latlnga);</pre>	Calculate distance to the first element
<pre>var distance;</pre>	Used to hold distance
<pre>for (i=1;i<content.length;i++) {</pre>	Set up iteration over content, starting at the second (index = 1) element
<pre>latlnga = new google.maps.LatLng(content[i][0],content[i][1]);</pre>	Build latitude/longitude object
<pre>distance = dist(clatlng,latlnga);</pre>	Calculate distance
<pre>if (distance < closestdistance){</pre>	Compare to the closest so far; if this one is less. ...
<pre>closestdistance = distance;</pre>	. . . then replace previous candidate for closest distance . . .
<pre>bestyet = i;</pre> and previous index

}	Close if true clause
}	Close for loop
distance = Math.floor((closestdistance+.005)*100)/100;	Set distance; note that formatting not needed at this stage; may choose to use in the future
if (v != undefined) {	Check if there was a previous video
v.pause();	Pause it
v.style.display = "none";	Remove from display
}	Close if previous video clause
if (audioel != undefined) {	Check if previous audio

audioel.pause();	Pause it
audioel.style.display = "none";	Erase controls for last audio played
}	Close if audio clause
ctx.clearRect(0,0,300,300);	Clear canvas

<pre>if (distance < maxdistance) {</pre>	Is this distance close enough?
<pre>marker = new google.maps.Marker({ position: clatlng, title: content[bestyet][2], icon: xmarker, map: map });</pre>	Mark the target position on the map; note that this is the target location, not the click location
<pre>switch (content[bestyet][3]) {</pre>	Determine what type of media this is
<pre>case "video":</pre>	For video
<pre>answer.innerHTML=content[bestyet][2];</pre>	Display the title
<pre>v = content[bestyet][4];</pre>	Set v to be the video
<pre>v.style.display="block";</pre>	Make the video element visible (The previous setting of style.display was " none ".)
<pre>v.currentTime = 0;</pre>	Set the video time to zero to start playback at the beginning; this restarts the video, just in case it has starting playing before

<code>v.play();</code>	Play the video
<code>break;</code>	Leave the switch statement
<code>case "picture":</code>	For picture
<code>case "pictureaudio":</code>	For pictureaudio
<code>answer.innerHTML=content[bestyet] [2];</code>	Display the title
<code>ctx.drawImage(content[bestyet] [4],10,10);</code>	Draw the image on the canvas
<code>if (content[bestyet][3]=="picture") {</code>	If it is picture . . .
<code>break;}</code>	. . . leave the switch
<code>else {</code>	Else (for the picture/audio combination)
<code>audioel = content[bestyet][5];</code>	Set audioel to be the audio element

audioel.style.display="block";	Display the audio—that is, the controls
audioel.currentTime = 0;	Set time to zero
audioel.play();	Play the audio
break;	Leave the switch
}	Close the picture or pictureaudio clause
}	Close the switch
}	Close the if close enough clause
else {	Else
answer.innerHTML="Not close enough to any [new] target";	Display message that click was not close enough to anything
}	Close clause

}	Close checkit function
function dist(point1, point2) {	Header for dist-between-two-points function
var R = 6371; // km	Factor used for kilometers (the radius of the earth)
// var R = 3959; // miles	Comment in code; leave in just in case you want to switch to miles
var lat1 = point1.lat()*Math.PI/180;	Change to radians
var lat2 = point2.lat()*Math.PI/180 ;	Change to radians
var lon1 = point1.lng()*Math.PI/180;	Change to radians

var lon2 = point2.lng()*Math.PI/180;	Change to radians
var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2-lon1)) * R;	Calculation based on spherical law of cosines

return d;	Return result
}	Close dist function
</script>	Closing script tag
</head>	Closing head tag
<body onLoad="init();">	Body tag, invoked init
<header id="header">Click on map.</header>	Header for the page
<div id="place" style="float:left; width:60%; height:400px"></div>	Set up div to float to the left; this will be the place for the map
<div style="float:right; width:38%; height:400px">	Set up div to float to the right; it holds the answer and the media (video, audio, and canvas)
<div id="answer">Title will be placed here.</div>	Place for the titles (i.e., text about the location and the media)
<p> </p>	Spacing
<video id="maze" preload="auto" controls="controls" width="400">	Video tag

<pre><source src="maze.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'></pre>	Possible source
<pre><source src="maze.theora.ogv" type='video/ogg; codecs="theora, vorbis"'></pre>	Possible source
<pre><source src="maze.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'></pre>	Possible source
Your browser does not accept the video tag.	Standard text for noncompliant browsers
<code></video></code>	Closing video tag

<pre><video id="fire" preload="auto" controls="controls"></pre>	Video tag
<pre><source src="sfire3.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'></pre>	Possible source
<pre><source src="sfire3.theora.ogv" type='video/ogg; codecs="theora, vorbis"'></pre>	Possible source
<pre><source src="sfire3.webmvp8.webm" type='video/webm; codec="vp8, vorbis"'></pre>	Possible source

Your browser does not accept the video tag.	Standard text for noncompliant browsers
</video>	Closing video tag
<audio id="mpiano" controls="controls" preload="preload">	Audio tag
<source src="estherT.ogg" type="audio/ogg" />	Possible source
<source src="estherT.mp3" type="audio/mpeg" />	Possible source
<source src="estherT.wav" type="audio/wav" />	Possible source
</audio>	Closing tag
<canvas id="canvas" width="300" height="300" >	Canvas tag
Your browser doesn't recognize canvas	Standard text for noncompliant browsers
</canvas>	Closing canvas tag
</div>	Closes div that floated from the right
</body>	Closing body tag
</html>	Closing html tag

The Mapmediabase Application

The next application, **mapmediabase.html**, recreates the first application, but uses a separate file for the content. You can easily make this one your own by changing the content. You need to obtain the latitude/longitude coordinates for your locations. To do so, you can use the **mapspotlight.html** application covered in the last chapter, you can use Google Maps directly, you can use Wolfram Alpha, or you can look them up in an atlas. A quick summary of the application follows:

1. **init**: Performs initialization, including the call to **loadcontent**.
2. **loadcontent**: Uses the variables, most significantly the **precontent** array included in the external script element, to create new markup for the media. It also invokes **makemap**.
3. **makemap**: Brings in the map and sets up event handling, including the call to **checkit**.
4. **checkit**: Compares the clicked location with the locations described in the **content** array. If one is close enough, then the associated media is shown and played.
5. **dist**: Computes the distance between two locations.

Table 5-4 outlines the functions in the second portal application. The

function table describing the invoked/called by and calling relationships for the **mapmediabase.html** application is similar for all the applications.

Table 5-4. Functions in the Second Portal Application

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the body tag	
makemap	Invoked by init	makemap
checkit	Invoked by addListener call in makemap	
dist	Invoked by checkit	dist
loadcontent	Invoked by init	

Table 5-5 shows the code for the second portal application, named **mapmediabase.html**. Please look back at the “External Script File” section to see the code (all the variable declarations) for the contents of **mediacontent.js**.

Table 5-5. Complete Code for the Second Portal Application

Code Line	Description

<!DOCTYPE html>	
<html>	
<head>	
<title>Clickable map </title>	
<meta charset="UTF-8">	
<style>	
header {font-family:Georgia,"Times New Roman",serif;	
font-size:20px;	
display:block;	
}	
video {display:none; position:absolute; top: 60px; right: 20px; }	Required to get video to be on the right
audio {display:none; position:absolute; top: 60px; right: 20px; }	Required to get audio control to be on the right
canvas {position:relative; top:60px}	
#answer {position:relative; font- family:Georgia, "Times New Roman", <u>Times</u> , <u>serif; font-size:16px;}</u>	
</style>	

<pre><script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js? sensor=false"> </script></pre>	
---	--

<pre><script type="text/javascript" src="mediacontent.js"> </script></pre>	Bring in the specific content
--	-------------------------------

<pre><script type="text/javascript" charset="UTF-8"></pre>	
--	--

Code Line	Description
<pre>var listener;</pre>	
<pre>var map;</pre>	
<pre>var myOptions;</pre>	
<pre>var ctx;</pre>	
<pre>var blatlng;</pre>	
<pre>var content = [];</pre>	
<pre>var answer;</pre>	

	<pre>var v;</pre>	
	<pre>var audioel;</pre>	
	<pre>var videotext1 = "<video id=\"XXXX\" preload=\"auto\" controls=\"controls\" width=\"400\"><source src=\"XXXX.mp4\" type='video/mp4; codecs='avc1.42E01E, mp4a.40.2'>";</pre>	The first dummy (template) text to be used to create markup for video element
	<pre>var videotext2=<source src=\"XXXX.theora.ogv\" type='video/ogg; codecs='theora, vorbis'><source src=\"XXXX.webmvp8.webm\" type='video/webm; codec='vp8, vorbis'>";</pre>	The second piece of dummy text to be used to create markup for video element
	<pre>var videotext3="Your browser does not accept the video tag.</video>";</pre>	The third piece of dummy text to be used to create markup for video element
	<pre>var audiotext1=<audio id=\"XXXX\" controls=\"controls\" preload=\"preload\"><source src=\"XXXX.ogg\" type='audio/ogg' />";</pre>	The first piece of dummy text to be used to create markup for audio element
	<pre>var audiotext2=<source src=\"XXXX.mp3\" type='audio/mpeg' /><source src=\"XXXX.wav\" type='audio/wav' /></audio>";</pre>	The second piece of dummy text to be used to create markup for audio element
	<pre>function init() {</pre>	

<code>ctx</code>	<code>=</code>
<code>document.getElementById("canvas").getContext('2d');</code>	
<code>answer = document.getElementById("answer");</code>	
Code Line	Description
<code>loadcontent();</code>	Invoke loadcontent to perform the rest of the initialization
<code>}</code>	
<code>function loadcontent() {</code>	Header for loadcontent function
<code> var divelement;</code>	Used to hold dynamically created div
<code> makemap(base[0],base[1]);</code>	Bring in the map
<code> var videomarkup;</code>	Used to hold the text string for the video element HTML markup
<code> var videoelementreference;</code>	Used to hold the reference to the newly created video element
<code> var audiomarkup;</code>	Used to hold the text string for the audio element HTML markup

<code>var audioelementreference;</code>	Used to hold the reference to the newly created audio element
<code>var imageobj;</code>	Used to hold the newly created Image object
<code>var name;</code>	Used to hold the name
<code>var savedimagefilename;</code>	Needed to hold the image file name because slot in inner array is overwritten
<code>for (var i=0;i<precontent.length;i++) {</code>	Iterate over the precontent items
<code>content.push(precontent[i]);</code>	Add the item (itself an array) to the content array

Code Line	Description
<code>name = precontent[i][4];</code>	Extract the name
<code>switch (precontent[i][3]) {</code>	Switch over the different type (video, picture, pictureaudio)
<code>case "video":</code>	For the video case
<code>divelement= document.createElement("div");</code>	Create a new container div

divelement.style = "float: right; width:30%;";	Style it to float to the right (though this does not work in all browsers)
videomarkup = videotext1+videotext2+videotext3;	Create the whole HTML markup for video elements
videomarkup = videomarkup.replace(/XXXX/g,name);	Change XXXX to the name given in precontent
divelement.innerHTML = videomarkup;	Set the constructed text to be the HTML within the div
document.body.appendChild(divelement);	Add the div to the body
videoelementreference = document.getElementById(name);	Obtain a reference to the created video element
content[i][4] = videoelementreference;	Put this value into the content array
break;	Leave switch
case "pictureaudio":	For the pictureaudio case
divelement = document.createElement("div");	Create a new container div
divelement.style = "float: right; width:30%;";	Style it to float to the right (though this does not work in all browsers)

```
audiomarkup = audiotext1+audiotext2;
```

Create the whole HTML markup for audio elements

Code Line	Description
audiomarkup = audiomarkup.replace(/XXXX/g,name);	Change the XXXX to the name given in precontent
divelement.innerHTML = audiomarkup;	Set the constructed text to be the HTML within the div
document.body.appendChild(divelement);	Add the div to the body
audioreference = document.getElementById(name);	Obtain a reference to the created audio element
savedimagefilename = content[i][5];	Save the file name
content[i][5] = audioreference;	Overwrite that position to be the reference to the audio
imageobj = new Image();	Create a new Image object

imageobj.src= savedimagefilename;	Set its src to be the file name
content[i][4] = imageobj;	Set the content item to be the Image object
break;	Leave the switch
case "picture":	In the case of picture
imageobj = new Image();	Create a new Image object
imageobj.src= precontent[i][4];	Set its src to be the file name
content[i][4] = imageobj;	Set the content item to be the Image object
break;	Leave the switch (not strictly necessary, because it is the last one, but leave in just in case new media types are added)
}	Close switch
}	Close for loop

```
}
```

Close **loadcontent** function

Code Line	Description
var xmarker = "x1.png";	Image for marker on map
function makemap(mlat,mlong) {	
var marker;	
blatlng = new google.maps.LatLng(mlat,mlong);	
myOptions = {	
zoom: zoomlevel,	Pick up this value from the external file
center: blatlng,	
mapTypeId: google.maps.MapTypeId.ROADMAP,	
};	
map = new google.maps.Map(document.getElementById("place"), myOptions);	
marker = new google.maps.Marker({	
position: blatlng,	
title: "center",	

<code>icon: xmarker,</code>	
<code>map: map});</code>	
<code>listener = google.maps.event.addListener(map, 'click', function(event) {</code>	
<code>checkit(event.latLng);</code>	
<code>});</code>	
<code>}</code>	
<code>function checkit(clatlng) {</code>	

Code Line	Description
<code>var i;</code>	
<code>var latlnga =new google.maps.LatLng(content[0][0],content[0] [1]);</code>	
<code>var bestyet = 0;</code>	
<code>var closestdistance = dist(clatlng,latlnga);</code>	

```
var distance;

for (i=1;i<content.length;i++) {

    latlnga = new
google.maps.LatLng(content[i][0],content[i]
[1]);

    distance = dist(clatlng,latlnga);

    if (distance < closestdistance) {

        closestdistance = distance;

        bestyet = i;

    }

}

distance =
Math.floor((closestdistance+.005)*100)/100;

if (distance<maxdistance) {

marker = new google.maps.Marker({
```

<code>position: clatlng, title: content[bestyet][2], icon: xmarker, map: map});</code>	
<code>if (v != undefined) {</code>	
<code>v.pause();</code>	
<code>v.style.display = "none";</code>	

Code Line	Description
<code>}</code>	
<code>if (audioel != undefined) {</code>	
<code>audioel.pause();</code>	
<code>audioel.style.display = "none";</code>	
<code>}</code>	
<code>switch (content[bestyet][3]) {</code>	
<code>case "video":</code>	
<code>answer.innerHTML=content[bestyet][2];</code>	
<code>ctx.clearRect(0,0,300,300);</code>	

<code>v = content[bestyet][4];</code>	
<code>v.style.display="block";</code>	
<code>v.currentTime = 0;</code>	
<code>v.play();</code>	
<code>break;</code>	
<code>case "picture":</code>	
<code>case "pictureaudio":</code>	
<code>answer.innerHTML=content[bestyet][2];</code>	
<code>ctx.clearRect(0,0,300,300);</code>	
<code>ctx.drawImage(content[bestyet][4],10,10);</code>	
<code>if (content[bestyet][3]=="picture") {</code>	
<code>break;}</code>	
<code>else {</code>	

Code Line	Description
<code>audioel = content[bestyet][5];</code>	
<code>audioel.style.display="block";</code>	

```
audioel.currentTime = 0;  
  
audioel.play();  
  
break;  
  
}  
  
}  
  
}  
  
else {  
  
    answer.innerHTML= "Not close enough to  
any [new] target.";  
  
}  
  
}  
  
function dist(point1, point2) {  
  
    var R = 6371; // km  
  
    // var R = 3959; // miles  
  
    var lat1 = point1.lat()*Math.PI/180;
```

<code>var lat2 = point2.lat()*Math.PI/180 ;</code>	
<code>var lon1 = point1.lng()*Math.PI/180;</code>	
<code>var lon2 = point2.lng()*Math.PI/180;</code>	
<code>var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2- lon1)) * R;</code>	
<code>return d;</code>	

Code Line	Description
<code>}</code>	
<code></script></code>	
<code></head></code>	
<code><body onLoad="init();"></code>	
<code><header id="header">Click on map.</header></code>	
<code><div id="place" style="float: left; width:50%; height:400px"></div></code>	
<code><div style="float: right; width:30%; height:400px"></code>	
<code><div id="answer">Title will be placed here.</div></code>	

```
<p> </p>
```

```
<canvas id="canvas" width="300" height="300" >
```

```
Your browser doesn't recognize canvas
```

```
</canvas>
```

```
</div>
```

```
</body>
```

```
</html>
```

The Quiz Application

The third and last application for this chapter was a quiz. The implementation was built on the second application. It makes use of an external script file. Look back at the “External Script File” section for the contents. A quick summary of the application follows:

1. **init**: Performs initialization, including the call to **loadcontent**.
2. **loadcontent**: Uses the variables, most significantly the **precontent** array included in the external script element, to create new markup for the media. It also invokes **makemap**. The **questions** array does not need any more work.
3. **makemap**: Brings in the map and sets up event handling, including the call to **checkit**.
4. **shufflecontent**: Shuffles the **content** and **questions** arrays (keeping them in correspondence).
5. **asknewquestion**: Displays the questions.
6. **checkit**: Compares the clicked location with the location for this question.
7. **dist**: Computes the distance between two locations.
8. **giveup**: Brings in a new map centered at the location for the current question.

9. **eraseold**: Removes any currently showing video, audio, or picture.

Table 5-6 outlines the functions in the quiz application. The function table describing the invoked/called by and calling relationships for the **mapmediabase.html** application is similar for all applications.

Table 5-6. Functions in the Quiz Application

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the <body> tag	loadcontent, asknewquestion
makemap	Invoked by init	
checkit	Invoked by addListener call in makemap	dist, asknewquestion
dist	Invoked by checkit	
loadcontent	Invoked by init	

asknewquestion	Invoked by init and checkit	
eraseold	Invoked by checkit and giveup	
giveup	Invoked by action of button	eraseold

Table 5-7 shows the code for the quiz application, with comments for new or changed statements.

Table 5-7. Include Table Caption

Code Line	Description
<!DOCTYPE html>	
<html>	
<head>	
<title>Map Quiz </title>	
<meta charset="UTF-8">	
<style>	
header {font-family:Georgia,"Times New Roman",serif;	

<code>font-size:20px;</code>	
<code>display:block;</code>	
<code>}</code>	
<code>video {display:none; position:absolute; top:60px; right: 20px;}</code>	
<code>}</code>	
<code>audio {display:none; position:absolute; top:60px; right: 20px;}</code>	
<code>canvas {position:relative; top:60px}</code>	
<code>#answer {position:relative; font-family:Georgia, "Times New Roman", <u>Times, serif</u>; font-size:16px;}</code>	
<code></style></code>	
<code><script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js?sensor=false"> </script></code>	
<code><script type="text/javascript" src="mediaquizcontent.js"> </script></code>	Bring in the content in mediaquizcontent.js

Code Line	Description
<script type="text/javascript" charset="UTF-8">	
var listener;	
var map;	
var myOptions;	
var ctx;	
var blatng;	
var content = [];	
var answer;	
var v;	
var audioel;	
var videotext1 = "<video id=\"XXXX\"	

<pre>preload=\"auto\" controls=\"controls\" width=\"400\"> <source src=\"XXXX.mp4\" type='video/mp4; codecs=\"avc1.42E01E, mp4a.40.2\"\'>;</pre>	
<pre>var videotext2=<source src=\"XXXX.theora.ogv\" type='video/ogg; codecs='theora, vorbis\'\' <source src=\"XXXX.webmvp8.webm\" type='video/webm; codec='vp8, vorbis\'\'>;</pre>	
<pre>var videotext3="Your browser does not accept the video tag.</video>;</pre>	
<pre>var audiotext1=<audio id=\"XXXX\" controls='controls' preload='preload'\><source src=\"XXXX.ogg\" type='audio/ogg'\>;</pre>	
<pre>var audiotext2=<source src=\"XXXX.mp3\" type='audio/mpeg'\><source src=\"XXXX.wav\" type='audio/wav'\></audio>;</pre>	
<pre>var nextquestion = -1;</pre>	The question counter needs to start before the 0th one
<pre>function init() {</pre>	
Code Line	Description
<pre>ctx = document.getElementById("canvas").getContext('2d');</pre>	
<pre>answer = document.getElementById("answer");</pre>	

header = document.getElementById("header");	
loadcontent();	
shufflecontent();	Invoke function to shuffle order of questions
asknewquestion();	Invoke function to ask question, thus starting off the quiz
}	
function shufflecontent() {	Header for shufflecontent function; two arrays will be shuffled
var i=content.length-1;	Start off at end
var s;	Hold the randomly

	designated spot
var hold;	For holding the content element, itself an array
var holdq;	For holding the question element
while(i>0) {	Start at the end and work down to zero
s = Math.floor(Math.random()*(i+1));	Pick a random position
hold = content[s];	Start with the content array; save the sth element
content[s]=content[i];	Swap in the ith element
content[i] = hold;	Put in the saved element

Code Line	Description
holdq = questions [<i>s</i>];	Now, working on the question array, save the <i>s</i> th element
questions [<i>s</i>]= questions [<i>i</i>];	Swap in the <i>i</i> th element
questions [<i>i</i>] = holdq ;	Put in the saved element
<i>i</i> --;	Decrement <i>i</i>
}	Close while loop
}	Close shufflecontent function
function asknewquestion () {	Header for asknewquestion function
nextquestion ++;	Increment the counter
if (nextquestion < questions.length) {	If still more questions
header.innerHTML = questions [nextquestion];	Show question
}	Close the if-still-more-question clause
else {	Else
header.innerHTML ="No more questions. ";	Display no more questions
}	Close else clause

<code>}</code>	Close asknewquestion function
<code>function loadcontent() {</code>	
<code> var divelement;</code>	
<code> makemap(base[0],base[1]);</code>	
<code> var videomarkup;</code>	
<code> var videoreference;</code>	

Code Line	Description
<code> var audiomarkup;</code>	
<code> var audioreference;</code>	
<code> var imageobj;</code>	
<code> var name;</code>	
<code> var savedimagefilename;</code>	
<code> for (var i=0;i<precontent.length;i++) {</code>	
<code> content.push(precontent[i]);</code>	
<code> name = precontent[i][4];</code>	
<code> switch (precontent[i][3]) {</code>	
<code> case "video":</code>	

<code>divelement= document.createElement("div");</code>	
<code>videomarkup = videotext1+videotext2+videotext3;</code>	
<code>videomarkup = videomarkup.replace(/XXXX/g,name);</code>	
<code>divelement.innerHTML = videomarkup;</code>	
<code>document.body.appendChild(divelement);</code>	
<code>videoreference = document.getElementById(name);</code>	
<code>content[i][4] = videoreference;</code>	
<code>break;</code>	
<code>case "pictureaudio":</code>	
<code>divelement = document.createElement("div");</code>	
<code>audiomarkup = audiotext1+audiotext2;</code>	
<code>audiomarkup = audiomarkup.replace(/XXXX/g,name);</code>	

Code Line	Description
<code>divelement.innerHTML = audiomarkup;</code>	
<code>document.body.appendChild(divelement);</code>	
<code>audioreference = document.getElementById(name);</code>	
<code>savedimagefilename = content[i][5];</code>	
<code>content[i][5] = audioreference;</code>	

<code>imageobj = new Image();</code>	
<code>imageobj.src= savedimagefilename;</code>	
<code>content[i][4] = imageobj;</code>	
<code>break;</code>	
<code>case "picture":</code>	
<code>imageobj = new Image();</code>	
<code>imageobj.src= precontent[i][4];</code>	
<code>content[i][4] = imageobj;</code>	
<code>break;</code>	
<code>}</code>	
<code>}</code>	
<code>}</code>	
<code>var xmarker = "x1.png";</code>	
<code>function makemap(mlat,mlong) {</code>	
<code> var marker;</code>	
<code> blatlng = new google.maps.LatLng(mlat,mlong);</code>	

Code Line	Description

<pre>myOptions = { zoom: zoomlevel, center: blatlng, mapTypeId: google.maps.MapTypeId.ROADMAP };</pre>	
<pre>map = new google.maps.Map(document.getElementById("place"), myOptions);</pre>	
<pre>marker = new google.maps.Marker({ position: blatlng, title: "center", icon: xmarker, map: map});</pre>	
<pre>listener = google.maps.event.addListener(map, 'click', function(event) {</pre>	
<pre> checkit(event.latLng); });</pre>	
}	
function eraseold() {	Header for eraseold function (same code as in previous example, but now in a function)
if (v != undefined) {	Is there an old v defined?
v.pause();	Pause it

<code>v.style.display = "none";</code>	Remove from display
<code>}</code>	Close clause
<code>if (audioel != undefined) {</code>	Is there an old audioel defined?
<code>audioel.pause();</code>	Pause it
<code>audioel.style.display = "none";</code>	Erase controls for last audio played
<code>}</code>	Close clause
<code>ctx.clearRect(0,0,300,300);</code>	Clear canvas
<code>}</code>	Close eraseold function

Code Line	Description
<code>function checkit(clatlng) {</code>	

<pre>var latlnga =new google.maps.LatLng(content[nextquestion][0],content[nextquestion][1]);</pre>	Build the latitude/longitude object for the answer to this question
<pre>var distance = dist(clatlng,latlnga);</pre>	Compute distance
<pre>eraseold();</pre>	Invoke the function to erase any media now on display
<pre>if (distance<maxdistance) {</pre>	Was the user's click close enough?
<pre>marker = new google.maps.Marker({position: latlnga, title: content[nextquestion][2], icon: xmarker, map: map});</pre>	Mark the correct location; function continues as in previous applications
<pre>switch (content[nextquestion][3]) {</pre>	
<pre>case "video":</pre>	
<pre>answer.innerHTML=content[nextquestion][2];</pre>	
<pre>ctx.clearRect(0,0,300,300);</pre>	

v = content[nextquestion][4];	
v.style.display="block";	
v.currentTime = 0;	
v.play();	
break;	
case "picture":	
case "pictureaudio":	
answer.innerHTML=content[nextquestion][2];	

```
ctx.clearRect(0,0,300,300);
```

Code Line	Description
ctx.drawImage(content[nextquestion][4],10,10);	
if (content[nextquestion][3]=="picture") {	
break;}	
else {	
audioel = content[nextquestion][5];	
audioel.style.display="block";	
audioel.currentTime = 0;	
audioel.play();	
break;	
}	
}	
asknewquestion();	Ask a new question (only if the user's guess was close enough)
}	
else {	

<code>answer.innerHTML= "Not close enough to the answer.";</code>	
<code>}</code>	
<code>}</code>	
<code>function dist(point1, point2) {</code>	
<code> var R = 6371; // km</code>	
<code> // var R = 3959; // miles</code>	
<code> var lat1 = point1.lat()*Math.PI/180;</code>	

Code Line	Description
<code> var lat2 = point2.lat()*Math.PI/180 ;</code>	
<code> var lon1 = point1.lng()*Math.PI/180;</code>	
<code> var lon2 = point2.lng()*Math.PI/180;</code>	
<code>var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2-lon1)) * R;</code>	
<code>return d;</code>	
<code>}</code>	
<code>function giveup() {</code>	Header for the giveup function

makemap(content[nextquestion][0],content[nextquestion][1]);	Bring in new map centered at the answer
eraseold();	Erase any old media
answer.innerHTML="Click at red x to finish this question.";	Display instructions since player needs to click to proceed this gives the player a way to indicate that he or she has seen the new map
}	
</script>	
</head>	
<body onLoad="init();">	
<header id="header">Click</header>	
<div id="place" style="float:left; width:50%; height:400px"></div>	
<button onClick="giveup();">Hint? </button>	Button indicated need for help
<div style="float: right; width:30%; height:400px">	

Code Line	Description
<div id="answer">Title will be placed here.</div>	
<p> </p>	
<canvas id="canvas" width="300" height="300" >	
Your browser doesn't recognize canvas	
</canvas>	
</div>	
</body>	
</html>	

Testing and Uploading the Application

I described three separate applications in this chapter. The **mapvideos.html** application consists of a single HTML file. The other two applications were each made up of two HTML files: one with the bulk of the coding and the other with the content. In all cases, the coding referenced the media. I used the same video files for the two video clips, the audio files for the single audio clip, and the two image files in all three cases. I used the image of a small hand-drawn *x* to mark locations on the map, instead of the default teardrop shape for markers in Google Maps.

As is the case for the project in the last chapter and the next chapter, this application does require you to be online to test since that is the only way to make contact with Google Maps.

Summary

In this chapter, you continued using the Google Maps API. You learned how to do the following:

- Program Google Maps API event handling to detect if the user was close to locations for which you had video, audio, or images
- Separate the definition of media content from the program itself
- Create HTML5 markup dynamically, using a regular expression to produce the right markup
- Start and stop the display and playing of media

In the next chapter, we will explore the use of geolocation, together with the Google Maps API, HTML5, and JavaScript and php to perform the sending of email.

C H A P T E R 6

Where am I: Using Geolocation, the Google Maps API, and PHP

In this chapter, you will learn the following:

- How to use geolocation to determine the location of your users/customers/clients
- About reverse geocoding in the Google Maps API to find the address of a given latitude/longitude
- About the asynchronous nature of certain operations in the Google Maps API
- How to send e-mail by using a server-side function

Introduction

The projects for this chapter involve the Google Maps API, geolocation, and PHP. I will describe two applications: the first just to show geolocation, a facility in which the browser determines the user's location using one of a variety of means. This application lets you find out where the location services think you are and compare it to where you really are. The second application provides a way for the user to generate an e-mail using the geolocation information along with other data and send it to someone. This makes use of a server-side PHP script to send the e-mail, and HTML5 form validation to confirm the e-mail addresses.

The geolocation specification is under development in parallel with HTML5 and is considered a distinct specification. It also is different from Google Maps, though Google Location Services is one of the main sources of geolocation information. An obvious thing to do with the location data is to display a Google Map, but you can do anything you want with the information.

One important feature of the geolocation specification is that the user—that is, the person on the client computer browsing the web page—must give approval for the location to be determined. This is so-called “opt-in.” The exact look of the screen varies with the different browsers. The terminology varies: a specific address may want to *know*, to *track*, or to *use*. It is all the same thing. Figure 6-1 shows the opening screen using Firefox.

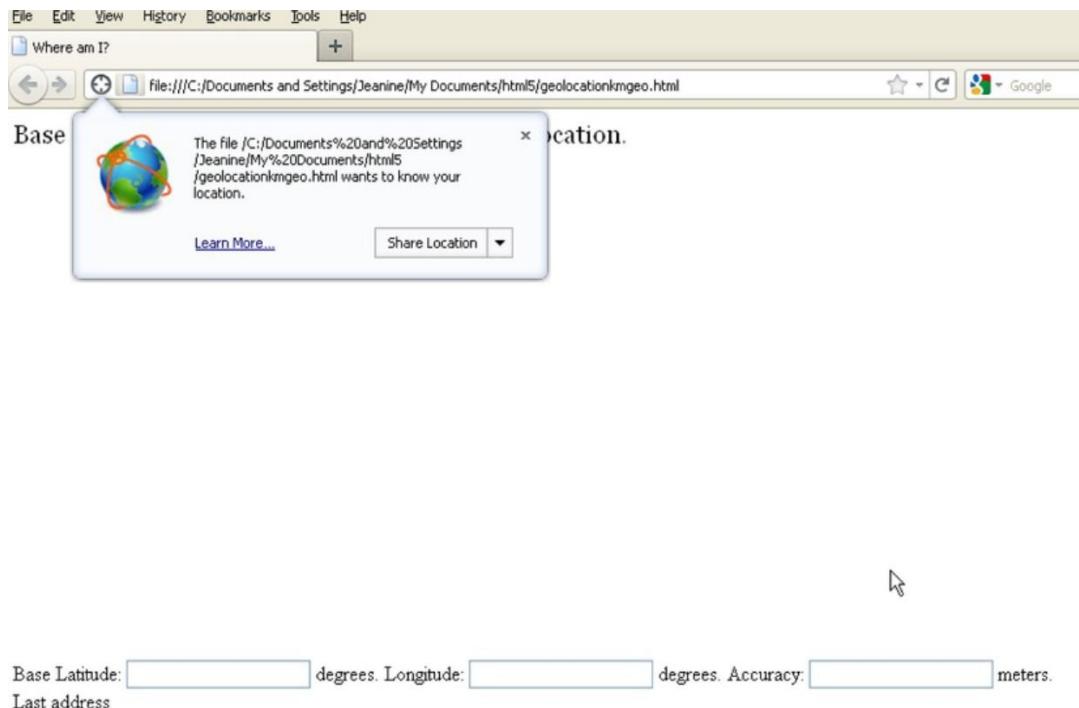
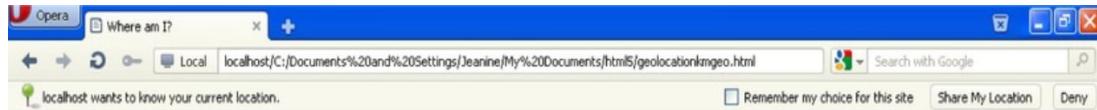


Figure 6-1. Opening screen with request for permission to share location (Firefox)

The drop-down menu offers four options: Share Location, Always Share, Never Share, and Not Now. If you choose the first option, when you refresh the screen or return to the page, the program will prompt again for a response.

Notice that other material already appears on the screen from my program. The request for permission is triggered when the code is invoked for the geolocation

operation. Figure 6-2 shows the analogous screen in the Opera browser.



Base location (small red x) is your current geolocation.

Base Latitude: degrees. Longitude: degrees. Accuracy: meters.
Last address

Figure 6-2. Opening screen with request for permission to share location (Opera)

Notice that Opera gives the option of remembering the choice for this site. Figure 6-3 shows a follow-on screen that requests another confirmation that the user is opting-in



Figure 6-3. Additional request from Opera

Figure 6-4 shows the corresponding screen in Chrome. At this point, I had uploaded the application to one of my server accounts. The browser uses the domain name of the account in its query.

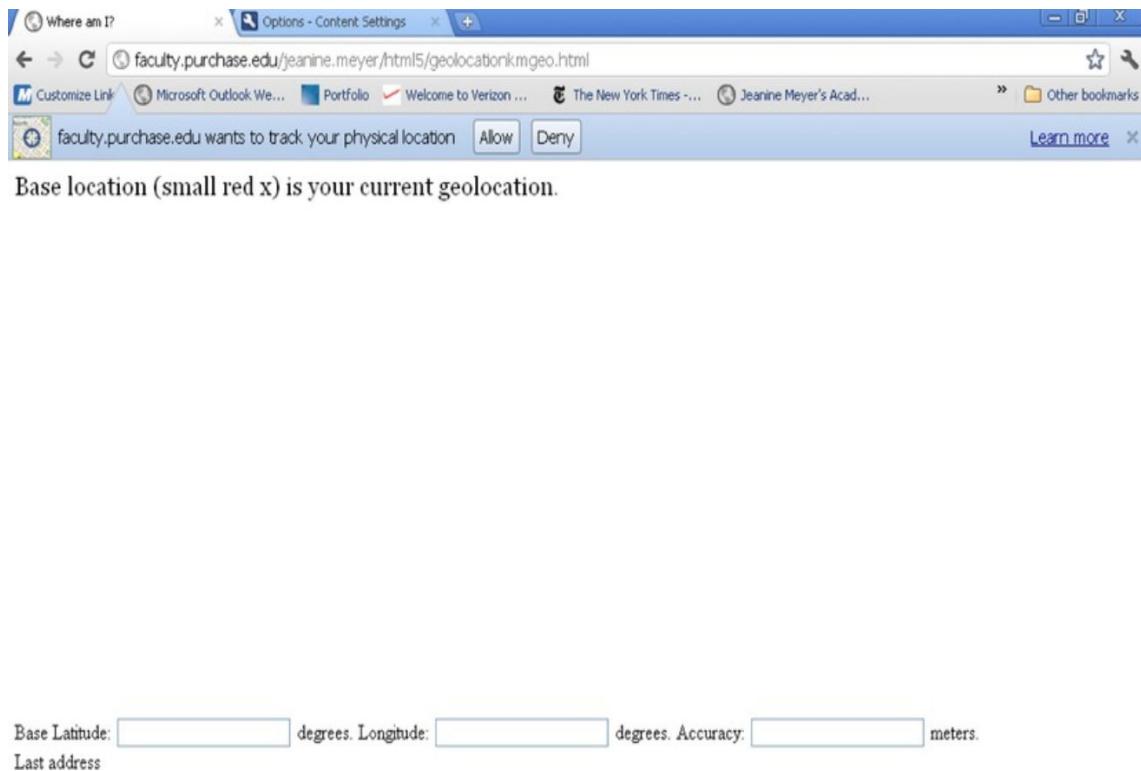
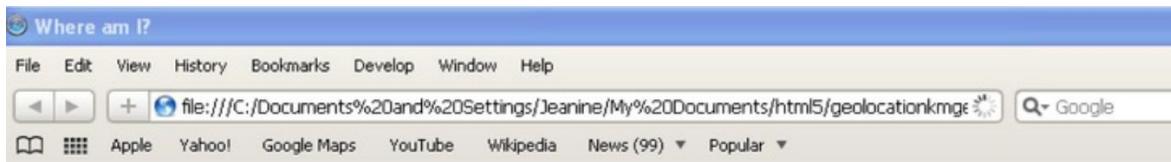


Figure 6-4. Opening screen with request for permission to share location (Chrome)

Figure 6-5 shows the screen for Safari. I returned again to using a file on my local computer.



Base location (small red x) is your current geolocation.

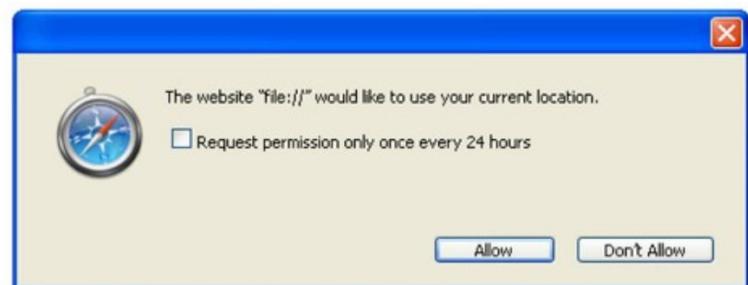
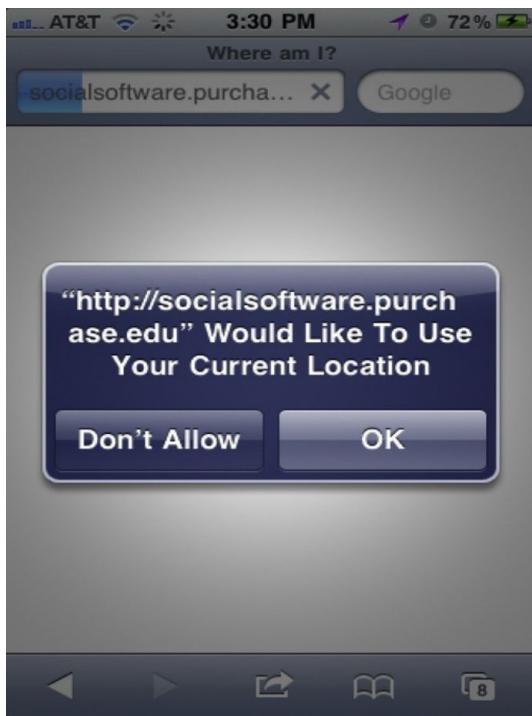


Figure 6-5. Request for permission to share location (Safari)



Notice that Safari provides the user a way to give permission for 24 hours—that is, avoid repeated requests each day. Note that Safari on my desktop PC does not work, and unfortunately hangs up—does nothing—rather than trigger the problem handler when the simplest call is made. I will show where it does work when explaining the different ways that geolocation actually is performed and techniques to use so that the problem handler will be triggered. For now, I note that the variant of Safari running on the iPhone does work. Figure 6-6 shows the permission screen for an iPhone. I wanted to test the other program, so I made use of another server account.

Figure 6-6. Request to share location (iPhone)

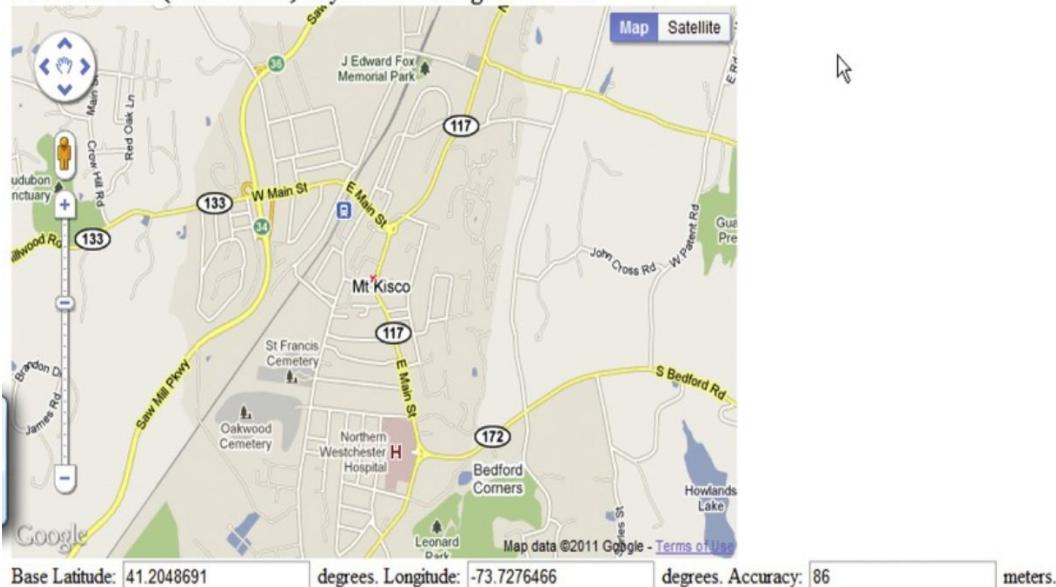
There are many variables, which will be explained following, but in my recent experiment around my town, the iPhone returned the most accurate results.

The permission is associated with the browser. For example, clicking “Remember my choice for this site” for Opera will not affect what happens when you use Firefox or any other browser. You also can use the general settings in the browser for all sites. For example, in Firefox, using Tools/Page Info/Permissions, you can choose from among Always Allow, Always Block, and Always Ask, the latter being what is recommended. In Opera, you can right-click (PC) or Ctrl+click (Mac), or choose Edit Site Preferences Network, to get the same choices. In Chrome, you start by clicking the wrench symbol in the upper right. [Then choose Options Under the Hood Content Settings, and scroll down](#) to Location. Safari appears to take a different approach. You can click the gear symbol in the upper right and then click Preferences and choose the Security tab. The choices are to allow sites to ask or not give permission at all. The iPhone provides similar options in Settings General Location Services.

The geolocation standard is moving its way through the recommendation process in the W3C (see [**www.w3.org/2008/geolocation/drafts/API/Implementation-Report.html**](http://www.w3.org/2008/geolocation/drafts/API/Implementation-Report.html)). You need to check and keep checking with each browser to determine what features work and how.

So, moving on, what does my program do after being given permission to determine the location? The program, [**geolocationkmgeo.html**](#), uses the returned coordinate values to bring in a Google Map, and uses another service, reverse geocoding, to calculate an address. Figure 6-7 shows the result. The geolocation is termed the base.

Base location (small red x) is your current geolocation.



Base Latitude: 41.2048691

degrees. Longitude: -73.7276466

degrees. Accuracy: 86

meters.

Last address

100-162 E Main St, Mt Kisco, NY 10549, USA

Distance

Figure 6-7. Location found in the basic program

The reverse geocoding has returned 100-162 E. Main Street, Mt. Kisco, NY 10549, USA for the description of the address with accuracy given as 86 meters. That is fairly accurate for the location of the red x. This screenshot was made while using a laptop at the Borders Café (now closed). That fact is significant because, as will be demonstrated, the geolocation itself was fairly accurate. The official W3C specification for geolocation supplies little

information on how the accuracy is calculated. This project can be used to make your own analysis of how accurate the geolocation is. The user can use the Google Maps interface to zoom and/or pan and then click the screen. This is what I did. A black dot will appear, along with the reverse-geocode address and the distance from the base of the clicked location. This can serve to check out distances to other locations, or calculate the distance from the geolocation to the actual location. Figure 6-8 shows the result of clicking the screen at where I determined I actually was.

Base location (small red x) is your current geolocation.

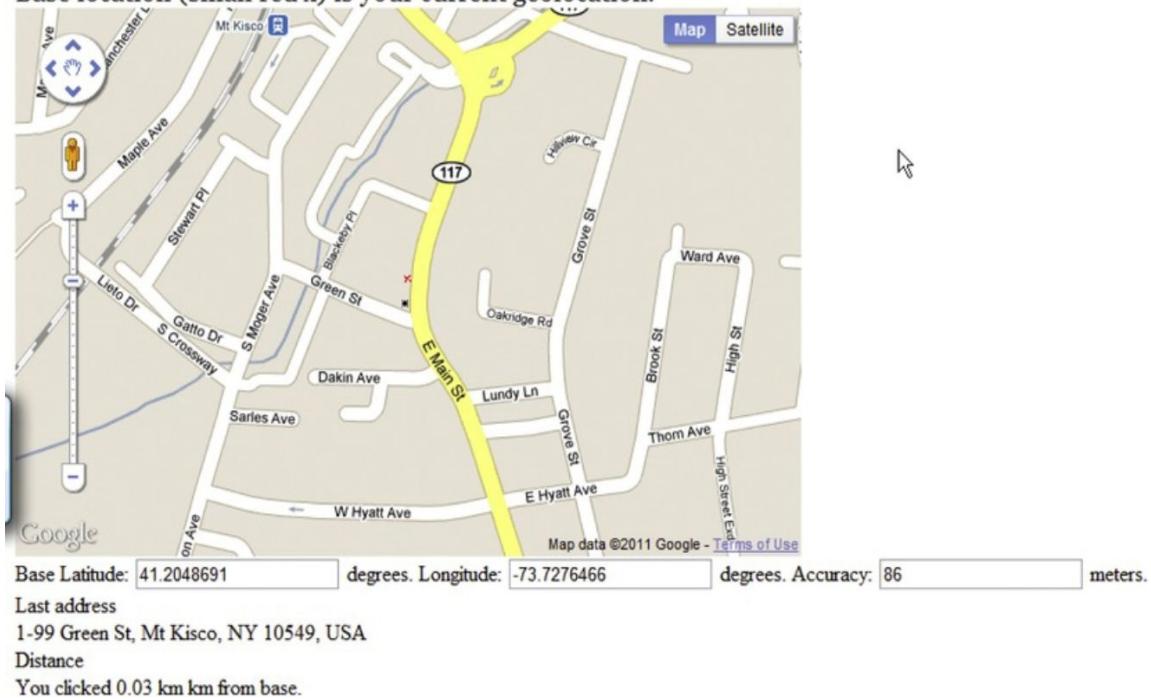


Figure 6-8. Screen showing actual location

The reverse geocoding is good: we were on Green Street. The program calculates and displays the distance from the base (the red x) and where I clicked (the black x). The location was .03 kilometers (equivalent to 30 meters). This is within the 86 meter accuracy returned by the geolocation function.

The second application demonstrates how you can use geolocation, reverse-geocode information, and actions you take yourself to compose an e-mail to send to someone else. Of course, many companies offer such services to facilitate meet-ups, promote restaurants, and so on. This application, **geolocationkme-mail.html**, makes use of a small program written in PHP that runs on your server to send e-mail to a person of your choosing. You will need to confirm that this is possible for your server account. This is an extra service that your Internet Service Provider may or may not provide. We'll will return to PHP in Chapter 10.

After agreeing to allow geolocation, the opening screen of the second application is shown (see Figure 6-9).

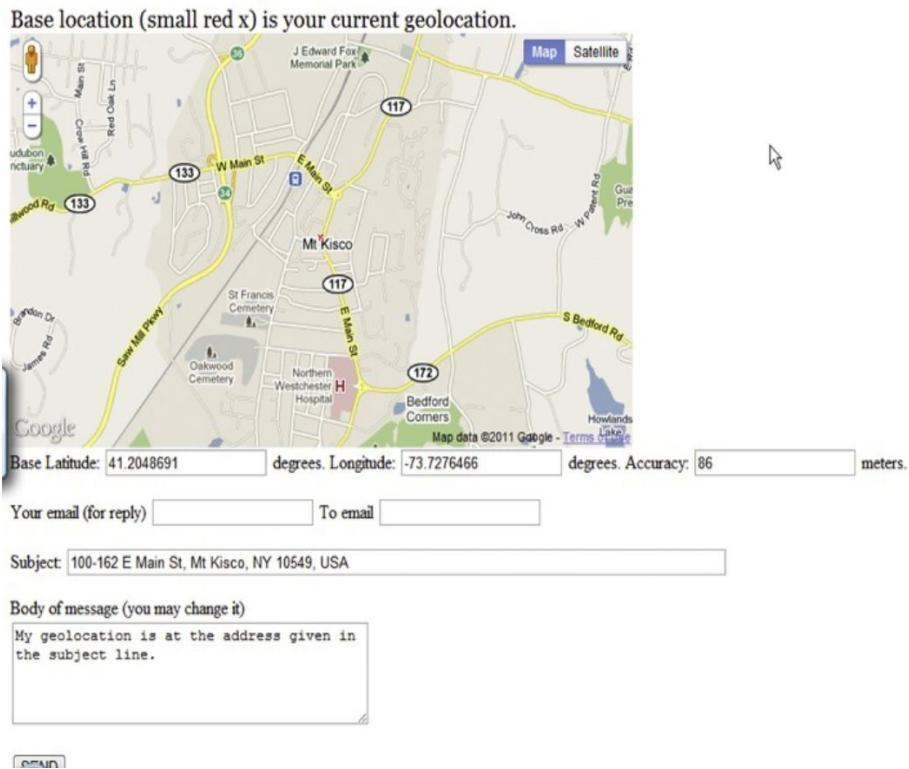


Figure 6-9. Opening screen for the e-mail program

Notice that there is a form on display with a place to put From and To e-mail addresses. The subject line has the reverse-geocoding information, and the body of the message refers to the subject line.

Next, I click where I believe I am. This application does require you to be able

to find yourself on a map! Figure 6-10 shows the results.

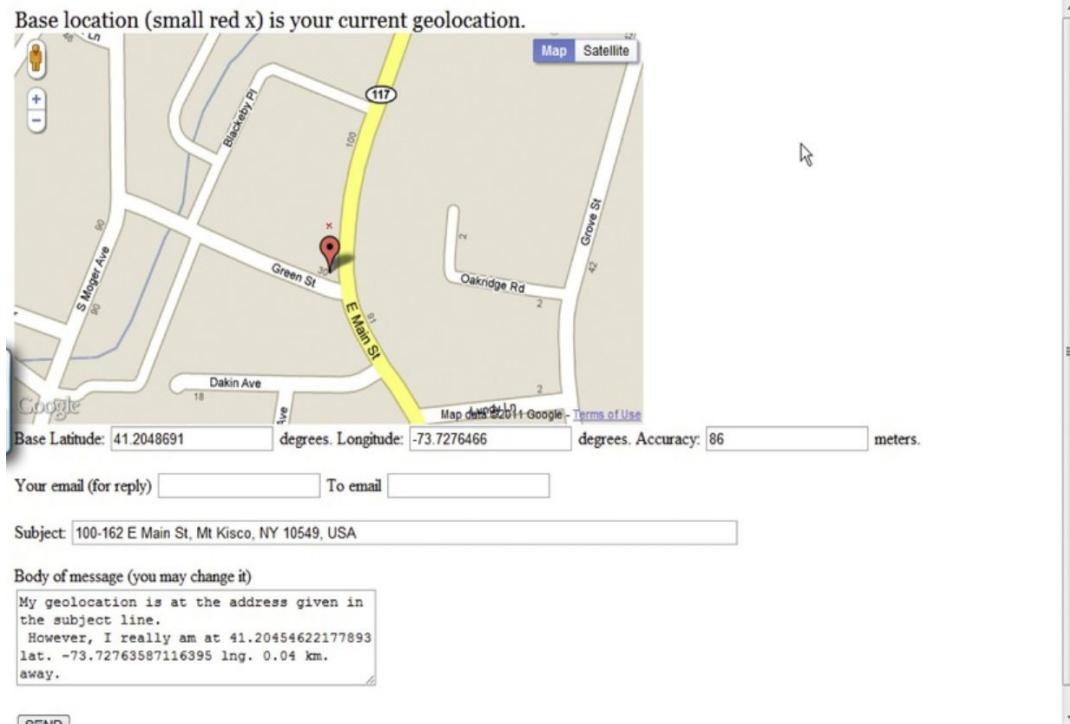


Figure 6-10. After clicking the location, the screen with information in the message body

The clicked location now has the default marker used by Google Maps. This actually was an oversight on my part, but I decided to stick with it to show you that you can use the default upside-down teardrop icon. Notice that the

subject line and the body of the e-mail have been filled. Now is the time to put in the From and To addresses. You can also change what is in the body of the e-mail or the subject line. You then click the SEND button. It is not instant, but the message will be sent to your e-mail account. Figure 6-11 shows the message as it appears in my Gmail account.



Figure 6-11. The received e-mail with location information

I am satisfied with how this works, but what I have shown you so far appears to assume that my users/customers/clients will be well behaved and put in proper e-mail addresses. This is a bad assumption to make. To handle this, I declared *form validation* as a requirement for this application. Form validation refers to a set of tests that are done to check if the input is valid. With form validation, if the user neglects to put in anything before hitting the SEND button, the program will present something like what is shown in Figure 6-12 (produced using Chrome).

Figure 6-12. Message from Chrome when a required field is empty

The screenshot shows a web form with two input fields: 'Your email (for reply)' and 'To email'. The 'Your email (for reply)' field is highlighted with a yellow border, indicating it is the current focus or has an error. A red speech bubble points to this field with the text 'Please fill out this field.' Below the form, the 'Subject' field contains the value '99 E Main St,' which is also highlighted with a yellow border.

If the user puts in something, but that something isn't a valid e-mail address, to the extent that it can be ascertained in terms of format alone, the application will present something like what is shown in Figure 6-13 (also produced using Chrome).

Figure 6-13. Invalid e-mail address as detected by Chrome

Other browsers also support similar form validation. Figure 6-14 shows the response produced by Firefox for an empty field.

Figure 6-14. Message from Firefox when a required field is empty

Figure 6-15 shows the response produced by Firefox for input that was not the correct format for e-mail.

Figure 6-15. Invalid e-mail address as detected by Firefox

One can argue that it would be better if all the form input fields were validated at once, but that could lead to an overcrowded screen. We hope the user gets the message. If not, the messages will be repeated for the To field. The user may wonder about the meaning of the From e-mail field. It does not mean that this message will show up in the SENT folder in the user's e-mail account. What it does is make it possible for the receiver to click Reply.

With this introduction to the projects for this chapter, I'll now provide background on geolocation and other critical requirements for these applications.

Geolocation and Other Critical Requirements

The main requirement for this application concerns geolocation: determining a latitude and longitude position for the client computer. In this section I will give a brief introduction to the wide variety of ways that the task may be accomplished for computers that range from the most mobile smartphone to the most sedentary desktop. The next section will describe the mechanics of how to use position information in HTML5 and JavaScript.

If you have a Global Positioning System (GPS) chip in your car, phone, or tablet computer, the notion of a computer program determining your location is commonplace. A GPS chip uses information from some of the 24 GPS satellites orbiting the earth to determine its location. A single satellite in the sky does not send down the information telling us where we are. Instead, the GPS device on the ground receives signals from multiple satellites. The device uses the current time and the specified time when each signal left the satellite to compute the travel time of each signal. It uses these travel times to compute its distances from the satellites and uses these distances along with the locations of the satellites to make the calculation. Using three satellites determines latitude and longitude; using four determines latitude, longitude, and altitude. Roughly speaking, the determinations are made by calculating the intersection of spheres: two spheres intersect in a circle. The software can make assumptions on the sphere that represents the planet earth, but it may not be accurate, as I know from one time using my car GPS and being “told” I was under the ramp to the Brooklyn Bridge. The local device may use signals from even more satellites, often up to ten, to confirm results. The best GPS applications use information from devices’ accelerometers to update their positions during brief interruptions in GPS signal reception, a trick that once was restricted to navigating submarines. A navigation device in a car or an app in a cell phone may also make use of data stored on the device to convert the

position into a street address.

So what happens if your computer does not have a GPS chip inside it, or is in a location from which it cannot receive satellite signals, but does have a cell phone radio and is in a cell phone service area? Another way to calculate position is to use the strengths of signals from one or more cell phone towers. Google Location Services and other geolocation applications have determined where many of the towers are and have stored that information in databases. If only a single cell phone tower is in range, a very rough position estimate is the location of that tower. However, if your computer can receive signals from several towers, then it can compute a much better estimate of your position. It uses the towers' different signal strengths instead of GPS satellites' different signal-travel times to compute your position by using known distances from known points.

If your computer has no GPS or cell phone radios but does connect by a short wire to a Wi-Fi radio, then the browser may be able to determine its position using the strengths of signals from other nearby Wi-Fi radios. It turns out that certain companies, such as Google Location Services, have databases with the known position of many, many Wi-Fi hot spots. The location of a Wi-Fi hot spot can be determined even if the hot spot is private and secure. The fact that Google, Apple, and probably other companies collect this information is controversial. One way of collecting the information has been through the vans that travel streets to get the images used for the Google Maps Street View service. Another way is to keep the data used when people use certain apps on their phones and forward it to the companies to add to their databases. In any case, these databases are referenced to provide hot spot positions for the geolocation calculations. Your computer can use the signal strengths of neighboring hot spots to estimate its distances from them, get their locations from the databases, and compute its own position. Of course, its

position then goes into the databases for other calculations to use!

Note Google, Apple, and others claim that data collected is stripped of personal information. I am not sympathetic to companies gathering information secretly. However, we do need to accept that many services are supplied for no explicit, per-use charge. To the adage/cliché, “you get what you pay for,” we may add something like, “The crowd or community bears a burden for what we don’t pay for.”

Lastly, what if your computer has no GPS, cell phone, or Wi-Fi radio? The IP address—the four-number combination that identifies your computer on the Web—is associated with a latitude and longitude value. It does not work well—the positions may be way off—if the computer shares an IP address with many others on a local network. However, this method allows the user of even an isolated computer to benefit from applications knowing the approximate position of the computer. My experience has been that the browsers on my desktop computer did not use Wi-Fi, but used the IP method, or, in the case of Safari, either failed or timed out (see the next section for more on the timeout option). In contrast, browsers on laptops and iPhones and iPads in my house, making use of the Wi-Fi network attached to my desktop computer, did make use of Wi-Fi for geolocation, with much better results.

These are the major technologies for determining position. The geolocation standard under development by the W3C establishes methods, attributes, and events for the use of programmers to access the information, if available.

My projects also make use of reverse geocoding, determining a street address or some sort of descriptive information about the position. Presumably, Google Location Services and other similar services have extensive databases on geographically descriptive terms associated with latitude and longitude values—not each value, certainly, but ranges of values. The HTML5 facility described in the next section provides several results produced by reverse geocoding, and you need to decide what is appropriate for your application. It will be dependent on the region of the country and the world.

Both projects require a response to the user clicking the map. This includes marking the position in a way that persists even when the map is moved or scaled. The projects also calculate the distance from the base location to the marked location.

Sending e-mail requires first of all a way for the user to enter the e-mail address. Sending e-mail is something beyond normal JavaScript processing, so to fulfill this requirement, it will be necessary to do something different—namely, to run a program on the server. Recall the basic mechanism of the Web is that files, including HTML documents, are located on server sites. A browser program, such as Firefox or Opera, runs on what is called the *client computer*—my computer or your computer in our house or at the coffee shop or library. The browser program fetches files from the server and interprets them. To perform a server operation such as sending e-mail, the browser must invoke a program running on the server.

Prior to sending the e-mail, the program needs to get the e-mail addresses, the To address to know where to send the message, and the From address so that reply will work. The program should do some sort of checking to make sure the addresses are valid. You will see that collecting and validating the information is done on the client computer, the one right in front of you or your user. Processing the e-mail and sending it are done by a PHP program running on the server, the computer holding the files for the project.

These are the requirements to build the projects demonstrated in the “Introduction” section. In the next section, I’ll describe the HTML5 and PHP features to implement the projects.

HTML5, CSS, JavaScript, and PHP Features

In this section, I will explain the features used to accomplish the requirements for the basic geolocation project and the e-mail project.

Geolocation

The W3C standard for geolocation is independent of how or what service actually supplies the information. It is assumed that the task will take time, so the request is set up as an asynchronous request. The code I wrote to make the request is

```
if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(handler, problemhandler, positionopts);  
}  
  
else {  
    if (document.getElementById("place")) {  
        document.getElementById("place").innerHTML = "I'm sorry but geolocation  
services are not supported by your browser";  
        document.getElementById("place").style.color = "#FF0000";  
    }  
}
```

The “Introduction” section showed successful examples of geolocation. At this point, I will show ways to handle problems, which can and do occur, before going on to explain how to use the information.

The condition for the outer **if** tests if **navigator.geolocation** is a recognized object for the browser. If it is *not*, then after checking that a div named **place** exists, the program displays the message starting with “I’m sorry . . .” When I tried this program in an old version of Internet Explorer, I first had to give permission to run any scripts, as shown in Figure 6-16.

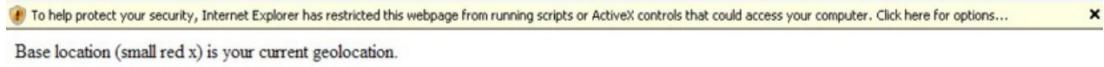


Figure 6-16. Internet Explorer request to run scripts

After I gave permission, Internet Explorer showed that geolocation was not available (see Figure 6-17).

Figure 6-17. Message on absence of geolocation in old Internet Explorer

If the **navigator.geolocation** object does exist, then my code invokes the **getCurrentPosition** method with three parameters. The first parameter, **handler**, is the name of a function that will be invoked if and when the operation is complete. The second parameter, **problemhandler**, is invoked when there is a problem. The third parameter, **positionopts**, is used to specify options. The setting I used for the options is

```
var positionopts;
```

```
positionopts = {  
    enableHighAccuracy: true,  
    timeout: 10000};
```

The interpretation of the **enableHighAccuracy** setting is not specified by W3C, but the practical implication for now is to use GPS if it is available. When enabled, more process time may be used, and the application as a whole may be slower. Setting this item to **false**, which is the default, also may preserve battery life on the local device. The timeout setting indicates that the time to perform this operation is to be limited to 10,000 milliseconds (10 seconds). There are other option settings, so when you start to use geolocation for production work, do investigate them.

If a problem is detected while performing the geolocation operation, the **problemhandler** function is invoked with a parameter containing a code indicating the nature of the problem. The definition of the **problemhandler** function includes a **switch** statement based on this code.

```
function problemhandler(prob) {  
    switch(prob.code) { case 1:  
        document.getElementById("place").innerHTML = "User declined to share the  
        location information.";  
        break;  
        case 2:  
        document.getElementById("place").innerHTML = "Errors in getting base location.";  
        break;  
        case 3:  
        document.getElementById("place").innerHTML = "Timeout in getting base location.";  
    }  
  
    document.getElementById("header").innerHTML = "Base location needs to be set!";
```

}

Base location needs to be set!
User declined to share the location information.

Base Latitude: degrees. Longitude: degrees. Accuracy: meters.
Last address
Distance

Figure 6-18

shows the result of the user denying permission to do geolocation.

Figure 6-18. Result of the user saying no to geolocation

A subtler problem is the failure to perform the operation in a timely manner. Figure 6-19 shows the situation alluded to earlier concerning using Safari on a desktop computer.

Figure 6-19. Geolocation taking too long

Let's move on now to the successful case: geolocation has worked. My code has caused the function handler to be invoked with the parameter set by the **getCurrentPosition** method to hold the calculated position information. The code extracts the latitude and longitude values and invokes my **makemap** function, which sets a global variable named **blatlng** and brings in the Google Map. The **makemap** function is essentially the same one used in the projects covered in the last two chapters. A Google Map with the full Google Map interface will appear on the screen. The handler function also displays values and calls **reversegeo**, the topic of the next section. Here is the code for **handler**:

```
function handler(position) {  
    var mylat = position.coords.latitude;  
    var mylong = position.coords.longitude;  
    makemap(mylat,mylong);  
    document.coutput.lat.value = mylat;  
    document.coutput.lon.value = mylong;  
    document.coutput.acc.value = position.coords.accuracy;  
    reversegeo(blatlng);  
}
```

The **accuracy** value is part of the W3C specification. Its exact meaning is not defined in the standard. One possibility is that it defines the radius of a circle around the returned point within which the actual position will lie. The value

returned when I use my desktop PC has been given as 22,000 meters, when in fact the actual location is only 610 meters from the returned location. In any case, , you probably would not share the accuracy information with your user if you were building a production application.

Note The `navigator.geolocation` object has other methods and properties. Most notably, there is a **watchPosition** method that sets up monitoring for changes in position or changes in how the position is calculated.

Reverse Geocoding

Reverse geocoding refers to a facility for obtaining description information, such as street addresses, from a latitude/longitude position. Keeping in mind garbage-in/garbage-out, if the original latitude/longitude is in error, then the address returned by a reverse-geolocation operation will also be in error.

The geocode facility I use is part of the Google Maps API, and includes features for obtaining long names, short names, postal codes, and so on. In the **init** function, my code constructs a Geocoder object and sets the variable **geocoder**:

```
geocoder = new google.maps.Geocoder();
```

The **reversegeo** function invokes the **geocode** method of this object. This is an asynchronous operation. That is, the method initiates an operation that takes time. The geocode method starts the task of determining the reverse geocoding. Control moves on to the next statement before the task is complete. Most asynchronous methods work by specifying a function to be invoked when the task is complete. The call to the **geocode** method designates a function. The way I designated the function was to define what is called an *anonymous function* in the method call itself. The function definition in its entirety is the second parameter of the call to **geocode**. The complete **reversegeo** function is shown next.

It essentially is one statement, the call to **geocode**, which contains within the call the definition of a function.

```
function reversegeo(latlng) {  
  geocoder.geocode({'latLng': latlng}, function(results, status) {  
    if (status == google.maps.GeocoderStatus.OK) {  
      if (results[0]) {  
        addressref.innerHTML = results[0].formatted_address;
```

```
    } else {
      alert("No results found");
    }
  } else {
    alert("Geocoder failed due to: " + status);
  }
});
```

The **geocode** method can be used to find locations using descriptions (addresses), or for reverse geocoding: determining descriptions from locations. The presence of the '**latlng**' coordinate is what indicates that this is a reverse-geocoding operation. My code specifies the latitude/longitude, and **geocode** returns descriptive information such as a street address.

Note If instead you specify an address using **{'address': locrequest}**, where **locrequest** is a string variable holding an address, then **geocode** will return the latitude/longitude. This provides a way for you to bring into your application the basic capability of Google Maps.

The **results** parameter is an array that holds the address information, starting with the most exact and working up. Figure 6-20 shows the alert box produced by this addition to the code:

```
var out = "";
for (var i=0;i<results.length;i++) {
  out += results[i].formatted_address +"\n";
}
```



```
alert(out);
```

Figure 6-20. The results array from geocoding

The reason the second and third lines are identical may have to do with the fact that Mt. Kisco is a coterminous village and town. In any case, you may need to run a program with this sort of display to determine what you want to use in your specific application. My examples use `results[0]`. You may decide it is best to use `results[3]`. It is up to you and can be viewed as a situation of precision vs. accuracy.

Clicking the Map

The requirements I have stipulated for these projects allow the user to click the map. As was discussed in the previous two chapters, the Google Maps API provides a way to set up event handling. In the **makemap** function, I include the line

```
listener = google.maps.event.addListener(map, 'click', function(event) {  
    checkit(event.latLng);  
});
```

The **checkit** function uses the clicked location to make a marker, calculate an address using **reversegeo**, calculate the distance from the base to the clicked location, and display information. In the basic geolocation application, the definition for **checkit** is the following:

```
function checkit(clatlng) {  
    var distance = dist(clatlng,blatlng);  
    var result;  
    distance = Math.floor((distance+.005)*100)/100;  
    var distanceString = String(distance);  
    marker = new google.maps.Marker({  
        position: clatlng,  
        title: distanceString,  
        icon: bxmarker,  
        map: map });  
    markersArray.push(marker);
```

```
reversegeo(clatlng);
distanceref.innerHTML = "You clicked "+distanceString+" km from base.";
}
```

The marker used is a custom one, a black x. The title, which is what you see if you move the mouse to the marker, is the distance, formatted to be a two-decimal number. The sentence starting with “You clicked . . .” is displayed as part of the body of the document.

The **checkit** function for the e-mail geolocation project is similar. The objective of this application is to compose an e-mail relating to the user’s location; look back to Figure 6-10. The definition of the **checkit** function for this application uses the default marker and creates a sentence about the distances for the body of the message.

```
function checkit(clatlng) {
var distance = dist(clatlng,blatlng);
var result;
distance = Math.floor((distance+.005)*100)/100;
var distanceString = String(distance)+" km. away.";
var newcoords = String(clatlng.lat())+" lat. "+String(clatlng.lng())+" lng.";
distanceString = newcoords+" "+distanceString;
marker = new google.maps.Marker({
position: clatlng,
title: distanceString,
map: map });
document.msg.body.value = document.msg.body.value + " However, I really
```

```
am at "+distanceString;
```

```
}
```

Checking E-mail Address Input and Invoking PHP to send e-mail

As described in the “Introduction” section, any application involving the sending of e-mails based on user input should attempt to check that the input fits the format for an e-mail address. Fortunately, this is one of the new features of HTML5. The input elements in forms have a **type** attribute, and one type is **e-mail**. Standard HTML (4 and earlier) provides a way to specify that submission of the form is to cause invoking of a file on the server. I describe this more in the next section, including the significance of the method setting. The complete form for e-mail follows:

```
<form name="msg" action="sendemailp.php" method="post">  
<p>Your email (for reply)  
<input type="email" name="from" required/>  
  
To email  
  
<input type="email" name="to" required />  
</p>  
  
Subject: <input type="text" name="subject" size="100" />  
  
<p>  
Body of message (you may change it) <br/>  
<TEXTAREA NAME="body" COLS=40 ROWS=5>  
  
My geolocation is at the address given in the subject line. </TEXTAREA>  
</p>  
<input type="submit" value="SEND" />  
</form>
```

This code produces the error checking shown in Figures 6-12 to 6-15 (shown previously), and should be appreciated as removing responsibilities from the programmer.

The action attribute in the form tag specified the php file sendemailp.php. This means that when the form is submitted, assuming the input is valid, the browser will send a message to the server (the computer from which it downloaded the HTML document). The message will be to invoke sendemailp.php. The input data will be passed along. I will now give a brief introduction to php. Chapter 10 on the database project will contain more information.

A Brief Introduction to the PHP Language

The sending of e-mail is a facility provided on servers, not client computers. There are several languages for writing what are called server-side scripts, and PHP is one of them. It is well-maintained and well-documented at www.php.net. You will read more about PHP in the description of a database project in Chapter 10.

The previous section showed how to invoke a PHP script as the result of submission of a form. The form input can be passed to the script in one of two ways: POST and GET. The GET way makes use of what is called the *query string*. The POST way is done using the HTTP headers. I chose to use POST for this example.

The main purpose of a PHP script is to do something on the server, probably using form input from an HTML document, and generate an HTML document to be passed back to the browser for display. It is not the case with the example for this project, but many PHP scripts consist of a mixture of HTML and PHP. The PHP portions are indicated by the delimiters `<?php` and `?>`. The function **echo** adds its input to the HTML document being created.

A feature of PHP is that variables, built in and programmer defined, start with dollar signs. For example, if the PHP script has been invoked by the action of a form, with the method specified as POST, and the form had an input element named `to`, then the line

```
$to = $_POST['to'];
```

would access the form input named `to` and assign it to the variable `$to`.

As mentioned, PHP scripts compose HTML documents, so a frequent operation is concatenation of strings. The operator in PHP for this is `.` (dot). The line

```
$headers = "From: " . $_POST['from'];
```

extracts the form input named `from` and adds it to the literal string `"From: "` to form a longer string, which is assigned to the variable

\$headers.

Another feature of PHP, which probably is the strangest to experienced programmers, is that variables can be included in strings. The statement

```
echo("There was a problem: <br>the body is $body,<br> the to is $to,<br> subject is$subject,<br> headers is $headers.");
```

produces the output shown in Figure 6-21.

There was a problem:

the body is My geolocation is at the address given in the subject line. ,
the to is jeanine.meyer@gmail.com,
subject is 99 E Main St, Mt Kisco, NY 10549, USA,
headers is From: jeanine.meyer@purchase.edu.

Figure 6-21. Demonstration of PHP echo of composed string

The values of the variables **\$body** , **\$to** , **\$subject** , and **\$headers** have been extracted and made part of the string. Notice also that the string contains the HTML markup **
**.

The sending of e-mail is fairly straightforward. There is a built-in PHP function, **mail**, that does the work, using as parameters the To address, the subject, the message body, and any header material. The function returns **true** if the sending operation was successful and **false** otherwise. Note that the sending operation can be successful and the recipient e-mail service may still reject the address as being nonexistent. The complete code for **sendemailp.php** is shown in the next section.

Building the Application and Making It Your Own

You can make these projects your own by combining the geolocation feature with more substantial applications. Knowing where the visitors to your site are located can help personalize the application, and perhaps influence the choice of images. An informal summary/outline of the basic geolocation application follows:

- **init**: For initialization, including invoking the call for geolocation, which is done asynchronously
- **handler and problemhandler**: For completing the geolocation
- **makemap**: For bringing in the Google Map
- **checkit**: For responding to clicks on the map and invoking **diste**
- **reversegeo**: For determining an address from a latitude/longitude value

Table 6-1 lists all the functions and indicates how they are invoked and what functions they invoke.

Table 6-1. Functions in the Basic Geolocation Project

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the body tag	
handler	Invoked by action of the getCurrentPosition call in init	makemap, reversegeo

reversegeo	Invoked in handler	
problemhandler	Invoked by action of the getCurrentPosition call in init	
makemap	Invoked by handler	
checkit	Invoked by action of addListener in makemap	dist
dist	Invoked by checkit	

Table 6-2 shows the code for the basic application, with comments for each line. Much of this code you have seen in the previous chapters.

Table 6-2. Complete Code for the Geolocation Project

Code Line	Description
<!DOCTYPE html>	Doctype header for HTML5
<html>	html tag
<head>	Head tag
<title>Where am I?</title>	Complete title element
<meta charset="UTF-8">	Meta tag for character sets
<style>	Style tag
header {font-family:Georgia,"Times New Roman",serif;	Set up formatting for header element, including font choices
 font-size:20px;	Font size
 display:block;	Set up line break before and after
}	Close style directive

Code Line	Description
</style>	Closing tag for style
<script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js?sensor=false"></script>	Script to bring in Google Map API
<script type="text/javascript" charset="UTF-8">	Opening script tag
var positionopts;	Set up global variable for options for geolocation
positionopts = {	Start definition of associative array
enableHighAccuracy: true,	Set request to try for high accuracy
timeout: 10000};	Set timeout limit
var addressref;	Will hold reference to address div element
var distanceref;	Will hold reference to distance div element
var headerref;	Will hold reference to header div element
var geocoder;	Will hold geocoder object
function init() {	Header for init function

<code>addressref = document.getElementById("address");</code>	Set addressref
<code>headerref = document.getElementById("header");</code>	Set headerref
<code>distanceref = document.getElementById("distance");</code>	Set distanceref
<code>geocoder = new google.maps.Geocoder();</code>	Create and set geocoder to be Geocoder object
<code>if (navigator.geolocation) {</code>	Does browser recognize navigator.geolocation ?

Code Line	Description
<code>navigator.geolocation.getCurrentPosition(handler, problemhandler, positionopts);</code>	If so, make the call with parameters as shown
<code>}</code>	Close clause
<code>else {</code>	Else
<code>if (document.getElementById("place")) {</code>	If there is a place with ID place
<code>document.getElementById("place").innerHTML = "I'm sorry but geolocation services are not supported by your browser";</code>	Set its contents to give message
<code>document.getElementById("place").style.color</code>	=

<code>"#FF0000";</code>	Turn contents red
<code>}</code>	Close clause
<code>}</code>	Close outer else clause
<code>}</code>	Close function
<code>var listener;</code>	Variable for listener (set but not used as variable)
<code>var map;</code>	Will be used to hold map
<code>var blatlng;</code>	Will hold the latitude/longitude object for the base
<code>var myOptions;</code>	Will hold the options used in bringing in a map
<code>function handler(position) {</code>	Header for handler function; if geolocation successful, it is invoked with the parameter position
<code> var mylat = position.coords.latitude;</code>	Set to the latitude

<code>var mylong = position.coords.longitude;</code>	Set to the longitude
<code>makemap(mylat,mylong);</code>	Invoke (my) makemap function
<code>document.coutput.lat.value = mylat;</code>	Display the latitude

Code Line	Description
<code>document.coutput.lon.value = mylong;</code>	Display the longitude
<code>document.coutput.acc.value position.coords.accuracy;</code>	= Display the accuracy
<code>reversegeo(blatlng);</code>	Invoke (my) function reversegeo
<code>}</code>	Close handler function

function reversegeo(latlng) {	Header for the reversegeo function
geocoder.geocode({'latLng': latlng}, function(results, status) {	Invoke the geocode method with a latLng object. The second parameter is an anonymous function. The function definition starts in this line and concludes 11 lines down, right before the closing bracket for the reversegeo function.
if (status == google.maps.GeocoderStatus.OK) {	If the status returned with a value equal to the google.maps.GeocoderStatus.OK value
if (results[0]) {	If results[0] exists
addressref.innerHTML = results[0].formatted_address;	Display this result
} else {	Otherwise
alert("No results found");	Issue alert message
}	Close clause

}	Close outer clause
else {	Else (the status was not OK)
 alert("Geocoder failed due to: " + status);	Issue alert message
}	Close clause

Code Line	Description
});	Close function definition, close Geocode method call
}	Close reversegeo function

function problemhandler(prob) {	Header for problemhandler function; prob will have information on the failure
switch(prob.code) {	switch statement based on prob.code
case 1:	A code of 1 is returned if the user declines to share location information.
document.getElementById("place").innerHTML = "User declined to share the location information.;"	Issue message
break;	Leave the switch
case 2:	A code of 2 is to be returned if errors are detected by the program performing the geocoding.
document.getElementById("place").innerHTML = "Errors in getting base location.;"	Issue message
break;	Leave the switch

case 3:	A code of 3 is to be returned if the geocoding takes too long. You can specify a time limit in the original call to the geocode method.
document.getElementById("place").innerHTML = "Timeout in getting base location.;"	Issue message
}	Close switch statement
document.getElementById("header").innerHTML = "Base location needs to be set!";	In all cases, set contents of header

Code Line	Description
}	Close problemhandler function
var rxmarker = "rx1.png";	Red x; used for the base
var bxmarker = "bx1.png";	Black x; used for clicks

function makemap(mylat,mylong) {	Header for makemap function
var marker;	Will hold marker
blatlng = new google.maps.LatLng(mylat,mylong);	Create a LatLng object and set the global variable blatlng
myOptions = {	Start setup of the associated array for options for the map
zoom: 14,	Set zoom to constant 14
center: blatlng,	Set the center of the map to be blatlng
mapTypeId: google.maps.MapTypeId.ROADMAP	Set the map type to be a ROADMAP . Other possibilities are SATELLITE , TERRAIN and HYBRID .
};	Close array
map = new google.maps.Map(document.getElementById("place"), myOptions);	Invoke the Map constructor method to bring in a map with the indicated options; set the map variable
marker = new google.maps.Marker({	Create a marker

<code>position: blatlng,</code>	Position blatlng
<code>title: "center",</code>	Title "center" This is text that will appear when you use the mouse to move the cursor near the marker.
<code>icon: rxmarker,</code>	Icon for marker set to be my custom rxmarker

Code Line	Description
<code>map: map});</code>	Marker is on map; close Marker method call
<code>listener = google.maps.event.addListener(map, 'click', function(event) {</code>	Set up event handling for clicking the map; function is defined here
<code> checkit(event.latLng);</code>	Function is the line checkit(event.latLng)
<code>});</code>	Close definition of function and call to addListener
<code>}</code>	Close makemap
<code>function checkit(clatlng) {</code>	Header for checkit function
<code> var distance = dist(clatlng,blatlng);</code>	Calculate distance from the position given by the input parameter to the base location held in blatlng
	Round off the distance to two

<code>distance = Math.floor((distance+.005)*100)/100;</code>	decimal places
<code>var distanceString = String(distance);</code>	Convert distance to a string
<code>marker = new google.maps.Marker({</code>	Create a marker
<code>position: clatlng,</code>	When the user clicks on the map ...
<code>title: distanceString,</code>	... make the title the formatted distance
<code>icon: bxmarker,</code>	... mark with a black x
<code>map: map });</code>	... mark map; close the array, close the call to Marker
<code>reversegeo(clatlng);</code>	Invoke reversegeo
<code>distanceref.innerHTML = "You clicked "+distanceString+" km from base.";</code>	Display distance information

Code Line	Description
<code>}</code>	Close checkit function
<code>function dist(point1, point2) {</code>	Header for distance between two points
<code>var R = 6371; // km</code>	Factor for kilometers
<code>// var R = 3959; // miles</code>	Factor for miles; keep in code just in case

<code>var lat1 = point1.lat()*Math.PI/180;</code>	Convert to radians
<code>var lat2 = point2.lat()*Math.PI/180 ;</code>	Convert to radians
<code>var lon1 = point1.lng()*Math.PI/180;</code>	Convert to radians
<code>var lon2 = point2.lng()*Math.PI/180;</code>	Convert to radians
<code>var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2- lon1)) * R;</code>	Calculation based on spherical law of cosines
<code>return d;</code>	Return distance
<code>}</code>	Close dist function
<code></script></code>	Closing script tag
<code></head></code>	Closing head tag
<code><body onLoad="init();"></code>	Body tag, including setting up call to init() ;
<code><header id="header">Base location (small red x) is your current geolocation.</header></code>	Header element
<code><div id="place" style="width:600px; height:400px"></div></code>	Div with ID place where Google Maps will go
<code><form name="coutput"></code>	Form tag
<code>Base Latitude: <input type="text" name="lat"> degrees. Longitude: <input type="text" name="lon"> degrees. Accuracy: <input type="text" name="acc"> meters.
</code>	Text and input elements (all used for output/display in this example)

Code Line	Description
</form>	Closing form tag
Last address <div id="address"></div>	Div for addresses
Distance <div id="distance"></div>	Div for distances
</body>	Closing body tag
</html>	Closing html tag

You can build on the basic geolocation program in many ways. For example, you can use the information along with the **dist** function in a way similar to what was demonstrated in the last chapter to do something based on how close the user was to any of a set of fixed locations. The e-mail geolocation project builds on the basic project in another way. It uses the information to send an e-mail to someone of the user's choice. An informal summary/outline of the e-mail geolocation application follows:

- **init**: For initialization, including invoking the call for geolocation, which is done asynchronously
- **handler and problemhandler**: For completing the geolocation
- **makemap**: For bringing in the Google Map
- **checkit**: For responding to clicks on the map and invoking **dist**
- **reversegeo**: For determining an address from a latitude/longitude value
- **sendemailp** (*a separate program to run on the server*): For performing the sending of e-mail

Table 6-3 lists all the functions and describes how they are invoked and what functions they invoke. The structure that has served us well in other situations is somewhat lacking now. The **sendemailp.php** function is invoked through

the **action** property setting in the second <form> tag in the body element:

```
<form name="msg" action="sendemailp.php" method="post">
```

Table 6-3. Functions in the E-mail Geolocation Project (Same As the Basic Geolocation Project)

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the <body> tag	
handler	Invoked by action of the getCurrentPosition call in init	
reversegeo	Invoked in handler	makemap, reversegeo
problemhandler	Invoked by action of the getCurrentPosition call in init	
makemap	Invoked by the handler	
checkit	Invoked by action of addListener in makemap	dist
dist	Invoked by checkit	

Table 6-4 shows the complete code for **geolocationkmemail.html**. New code statements and changed code statements have comments.

Table 6-4. Code for the E-mail Geolocation Application

Code Line	Description

<!DOCTYPE html>	
<html>	
<head>	
<title>Where am I?</title>	
<meta charset="UTF-8">	
<style>	
header {font-family:Georgia,"Times New Roman",serif;	
font-size:20px;	
display:block;	
}	
</style>	

Code Line	Description
<script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js?sensor=false"> <td data-bbox="1090 1157 1429 1294"></td>	
<script type="text/javascript" charset="UTF-8">	
var positionopts;	
positionopts = {	
enableHighAccuracy: true,	
timeout: 10000} ;	
var headerref;	
var geocoder;	

function init() {	
headerref = document.getElementById("header");	
geocoder = new google.maps.Geocoder();	
if (navigator.geolocation) {	
navigator.geolocation.getCurrentPosition(handler,	
problemhandler, positionopts);	
}	
else {	
if (document.getElementById("place")) {	
document.getElementById("place").innerHTML = "I'm	
sorry but geolocation services are not supported by your	
browser";	
document.getElementById("place").style.color = "#FF0000";	
}	
}	

Code Line	Description
}	
var listener;	
var map;	
var blatlng;	
var myOptions;	
function handler(position) {	

var mylat = position.coords.latitude;	
var mylong = position.coords.longitude;	
var result;	
makemap(mylat,mylong);	
document.output.lat.value = mylat;	
document.output.lon.value = mylong;	
document.output.acc.value = position.coords.accuracy;	
reversegeo(blatlng);	
}	
function reversegeo(latlng) {	
geocoder.geocode({'latLng': latlng}, function(results, status) {	
if (status == google.maps.GeocoderStatus.OK) {	
if (results[0]) {	
document.msg.subject.value = results[0].formatted_address;	Put value in the subject input element of the message form

Code Line	Description
} else {	
alert("No results found in reverse geocoding.");	
}	
}	
else {	
alert("Geocoder failed due to: " + status);	
}	

});	
}	
function problemhandler(prob) {	
switch(prob.code) {	
case 1:	
document.getElementById("place").innerHTML = "User declined to share the location information.;"	
break;	
case 2:	
document.getElementById("place").innerHTML = "Errors in getting base location.;"	
break;	
case 3:	
document.getElementById("place").innerHTML = "Timeout in getting base location.;"	
}	
Code Line	Description
document.getElementById("header").innerHTML = "Base location needs to be set!";	
}	
var xmarker = "x1.png";	
function makemap(mlat,mlong) {	
var marker;	
blatlng = new google.maps.LatLng(mlat,mlong);	

myOptions = {	
zoom: 14,	
center: blatlng,	
mapTypeId: google.maps.MapTypeId.ROADMAP	
};	
map = new google.maps.Map(document.getElementById("place"), myOptions);	
marker = new google.maps.Marker({	
position: blatlng,	
title: "center",	
icon: xmarker,	
map: map});	
listener = google.maps.event.addListener(map, 'click', function(event) {	
checkit(event.latLng);	
});	

Code Line	Description
}	
function checkit(clatlng) {	
 var distance = dist(clatlng,blatlng);	
 var result;	
 distance = Math.floor((distance+.005)*100)/100;	

<code>var distanceString = String(distance)+" km. away.;"</code>	
<code>var newcoords = String(clatlng.lat())+" lat. "+String(clatlng.lng())+" lng.;"</code>	Used as part of message body
<code>distanceString = newcoords+" "+distanceString;</code>	Used as part of message body
<code>marker = new google.maps.Marker({</code>	Note that default icon is used
<code> position: clatlng,</code>	
<code> title: distanceString,</code>	
<code> map: map});</code>	
<code>document.msg.body.value = document.msg.body.value + " However, I really am at "+distanceString;</code>	Add to message body
<code>}</code>	
<code>function dist(point1, point2) {</code>	
<code> var R = 6371; // km</code>	
<code> // var R = 3959; // miles</code>	
<code> var lat1 = point1.lat()*Math.PI/180;</code>	
<code> var lat2 = point2.lat()*Math.PI/180 ;</code>	
<code> var lon1 = point1.lng()*Math.PI/180;</code>	

Code Line	Description
<code>var lon2 = point2.lng()*Math.PI/180;</code>	
<code>var d = Math.acos(Math.sin(lat1)*Math.sin(lat2) + Math.cos(lat1)*Math.cos(lat2) * Math.cos(lon2- lon1)) * R;</code>	

<pre>return d;</pre>	
<pre>}</pre>	
<pre></script></pre>	
<pre></head></pre>	
<pre><body onLoad="init();"></pre>	
<pre><header id="header">Base location (small red x) is your current geolocation.</header></pre>	
<pre><div id="place" style="width:600px; height:350px"></div></pre>	
<pre><form name="coutput"></pre>	
<pre>Base Latitude: <input type="text" name="lat"> degrees.</pre>	
<pre>Longitude: <input type="text" name="lon"> degrees.</pre>	
<pre>Accuracy: <input type="text" name="acc"> meters.
</pre>	
<pre></form></pre>	
<pre><form name="msg" action="sendemailp.php" method="post"></pre>	Form tag with specification of the action attribute causing sendemailp.php to be invoked, and specification of the method attribute causing the call to be made by the POST method
<pre><p>Your email (for reply)</pre>	Label for the From input field
<pre><input type="email" name="from" required/></pre>	This field is required and is of type email

The e-mail application requires a PHP script to do the actual work of sending the e-mail. My **sendemailp.php** program is shown in Table 6-5.

Table 6-5. Code for the PHP Script for Sending E-mail

Code Line	Description
<?php	Delimiter for PHP code
\$to = \$_POST['to'];	Set the variable \$to to the value of the form input with name to
\$subject = \$_POST['subject'];	Set the variable \$variable to the value of the form input with name subject
\$body = \$_POST['body'];	Set the variable \$body to the value of the form input with the name body
\$headers = "From: " . \$_POST['from'];	Set the variable \$headers to be a string starting with From: followed by the value of the form input with name from
if (mail(\$to, \$subject, \$body,\$headers)) {	Invoke the mail function, if it works
echo("Your message was sent");	Output success message
} else {	Else
echo("There was a problem: the body is \$body, the to is \$to, subject is \$subject, headers is \$headers.");	Output longer message, with the contents of all the variables
}	Close clause
?>	End the PHP

You may find uses for sending e-mail for other applications besides ones using geolocation. You also can build on this application in other more substantial ways, such as including other information in the e-mail content.

If you have an application in which you do not want to depend on geolocation exclusively, you may consider using a form in which the user can type in an address. You can use **geocoder.geocode** with the address parameter to obtain the latitude/longitude and bring in a map using that value. Another alternative is to have a list of places such as I presented in Chapter 4.

Testing and Uploading the Application

You need to be online to test the first application since that is the only way to make contact with Google Maps, but the HTML file and the files for the marker icons (**bx1.png** and **rx1.png**) can be on your local

computer. To test the e-mail application, you need to run the program from your server. That is, you need to upload all the files to the server. My files are **geolocationkmemail.html**, **rx1.png**, and **sendemailp.php**. Moreover, I'll repeat what I mentioned earlier: you need to check that the server allows PHP and allows PHP to send e-mail. To make this concrete, for example, my standard server at my college allows some use of PHP, but not e-mail. The local IT group set up a special server for my database course that does the job!

Summary

In this chapter, you continued using the Google Maps API. You learned how to use the following:

- Geolocation
- Google Maps API and geocoding for addresses
- A PHP script to do e-mail

In the next chapter, we leave geography for the smaller but still spatially fascinating world of paper folding. We will explore how to produce directions for an origami of a talking fish using JavaScript,

Index

A Accumulator, 303

addEventListener, 142 addListener function, 111 Adobe Flash, 332

B

Bouncing video, 53 animation
automatic scrolling, 63 clearInterval(tid), 61 ctx.clearRect(0,0,cwidth,cheight), 61 displacement value, 62
init function, 61 moveandcheck function, 61, 62 setInterval(drawscene,50), 61 setInterval function, 61
tid = setInterval(drawscene,50), 61 videobounceC program, 62 videobounceE program, 62, 63 video element bouncing with less
restrictive checking, 63 application
changedims function, 86 testing and uploading, 86 trajectory function, 69 v.currentTime attribute, 86
videobounceC application code, 69, 70
videobounceE program code, 76 VideobounceTrajectory program
code, 82 window.innerWidth and
window.innerHeight attributes, 86

body definition and window dimensions
ballrad variable, 61 body element, 60 init function, 60
Math.min method, 60 video element, 59 video formats, 59
window.innerWidth and window.innerHeight attributes, 59
HTML5, 53 looping video, 66
movable video element, 65, 66 Opera screen capture, 53, 54 project history and critical requirements, 58, 59 smaller window, 55, 56
stop-motion photography, 54 trajectory of virtual ball, 54, 55 traveling mask, 66–68
user interface, 68
very small window, 57
video drawn on canvas, 64, 65
window resize, running program, 57, 58

C

changescale function, 10 checkpositions function, 340 clearshadow function, 110 Corel Paint Shop Pro, 330 Crease pattern, 228, 229

D Database

building application, 390
adding and removing site records, 404
code for createresearchtables.php, 392
displaying information on sites, 417 end-users and database administrators, 390
finder registration scripts, 394 tasks by scripts, 390
client side vs. server side processing, 382, 383
critical requirements, 376 finders, 367
adding site, 371, 372 change password, 370, 371 delete site, 374, 375
list of web sites, 372, 373 registration, 368
user error, 369 hash function, 382 local storage, 381
Opera browser, 375, 376 PHP (see PHP)
relational databases, 377, 378 SQL, 379, 381
testing and uploading, 422
Database management system (DBMS), 377, 378
Document object model (DOM), 59 dologo function, 9, 11
Drop LatLng marker option, 100, 103

E

Entity relationship diagram (ERD), 377

F

Family collage, 19 Adobe Photoshop, 21
critical requirements, 21 canvas element, 22
drag and drop operation, 22 CSS, JavaScript features, 22 end-user position, 19
final product, rearranged objects, 20, 21 HTML5, 22

image application, 35
event handling functions, 35 HTML5 Family Card project, 35, 36 HTML5 Logo project, 37–52
initialization, 35
object definition methods, 35 JavaScript object, 22
constructor function, 22 family picture project, 22 method function, 22 types of objects, 22
manipulating object, 19 mouse over object, 28
coordinate system, 29 outside function, 30 overcheck method, 28 overheart function, 29 overoval function, 29 overrect function, 28
startdragging and makenewitem, 28 opening screen, family pictures, 20 save canvas image, 34
DataURL, 34 Firefox browser, 34
saveasimage function, 34 test and upload application, 52 user interface, 31
clone function, 32 drawstuff function, 31 dropit function, 34 flypaper effect, 33
mouse cursor coordinates, 32 moveit function, 33
onClick attributes, 31 removeobj function, 31
fillStyle property, 7 Frames, 64

G

Geolocation, 183

accuracy value, 200 Always Share option, 184 application
e-mail geolocation application code, 215
188 getCurrentPosition method, 198 handler function, 200
Internet Explorer scripts run request, 196, 197
invalid e-mail address Chrome, 193 Firefox, 193

iPhone permission screen, 187, 188 makemap function, 200, 202 message body information, 192 message on absence, 197

navigator.geolocation object, 198, 200 Never Share option, 184

Not Now option, 184 opening screen

Chrome, 186

e-mail program, 191 Firefox, 183, 184

Opera, 184, 185 Safari, 186, 187

Opera follow-on screen, 185 opt-in, 183

PHP

 checking e-mail address input, 203, 204

 language, 204, 205 positionopts, 198 problemhandler, 198 received e-mail with location information, 192 reverse geocoding, 189, 190

 anonymous function, 200 array, 201, 202

 definition, 200 Google Maps API, 200 init function, 200 locrequest, 201

 reversegeo function, 200 Share Location option, 184 specification, 183

 subtler problem, 199

 user denying permission, 198, 199 W3C recommendation process, 188 W3C standard, 196

 Global Positioning System (GPS), 194 Google Location Services, 183, 194 Google Maps API addListener, 104 associative array, 105 HYBRID map, 105 makemap function, 105

 Map constructor method, 104 Map, LatLng, and Marker, 104 map portal

 associative array, 138 event handling, 141

 HTML document location, 138 HYBRID map type, 139 interface removed, 141

 latitude and longitude values, 138 makemap function, 138 myOptions array, 140

 SATELLITE map type, 140

 TERRAIN map type, 138, 139 x1.png file, 141

mobile devices applications, 104 onLoad attribute, 104

portal construction, 105 pseudocode, 104 ROADMAP, 105 SATELLITE map, 105 TERRAIN map, 105

H

HTML5 logo, 1

body of document, 7, 8 Building section, 4 canvas element, 1

Chrome browser opening screen, 2 coordinate transformation, 8, 9 drawing paths

canvas element, 5 2D context, 5

2D coordinate system, 6 closePath method, 6 hexadecimal format, 6 init function, 6, 7 onLoad attribute, 6 sequence, 5

drawpath, fillStyle property, 5 Firefox opening screen, 3 graceful degradation, 3 implementation, 3

project code, 12 project function, 11

project history and critical requirements, 4, 5

range input element, 9–11 scaled down, 3

semantic tags, 1 slider feature, 2

testing and uploading, 18 Test section, 4

text placement, 7, 8

World Wide Web Consortium, 2 HYBRID map, 139

I

initMouseEvent method, 302 innerHTML attribute, 335 intersect function, 239 Intersection, 240

J

JavaScript arrays, 147

JavaScript object, constructor function drawing, 26, 28

heart, 25 Oval, 23, 24 picture, 24, 25 Rect, 23

Jigsaw video puzzle, 283 application

jigsaw-to-video project code, 307 jigsaw-to-video project functions, 306

testing and uploading, 319 desktop computer

Feedback label, 286

nearly completed puzzle, 286, 287 opening screen, 283, 284

puzzle progress, 285, 286 replaced pieces, 288 spread out pieces, 284, 285 tolerance, 287

video with controls, 288, 289 display attribute, 297 endjigsaw function, 296

finger touches accumulator, 303

checkpositions function, 303, 304 deltax and deltay arrays, 303 doaverage function, 303

piecesx and piecesy arrays, 303 questionfel element, 304 release function, 303 setupjigsaw function, 302

tolerance, 303

touchcancel, 302 touchend, 302

touchHandler code, 302, 303 touchmove, 302

touchstart, 302

video preparation, positioning, and playing, 305

W3C, 302

firstpkel variable, 296

images and data acquisition, 294, 295 init function, 295

iPhone and iPad

critical requirements, 293 game in progress, 291

jigsaw-puzzle-with-video-reward project, 293

opening screen, 289, 290 ready to play video, 291, 292

user interface construction, 289 video in play, 293

Math.floor, 297 Math.random, 297 mouse events

adjustX, 299, 300 adjustY, 299
checkpositions function, 301 curX, 298–300
curY, 298
draw function, 301 Internet Explorer, 298
mouseDown variable, 300, 301 moving function, 300
moving jigsaw pieces, 298 movingobj element, 301 movingobj variable, 300 offset function, 300
pieceelements array, 298
setupjigsaw function, 298, 301 startdragging function, 298–301 style element, 301
zero offset, 300
piecesx and piecesy values, 296 piecesx file, 296
setupgame function, 295, 296 setupjigsaw function, 296, 297

K

kamih variable, 237 kamiw variable, 237

M

Map maker, Google Maps, 89 API (see Google Maps API) application functions, 114, 115 mapspotlight.html application code, 115 testing and uploading, 127 base location, 90, 93–95 canvas graphics drawshadowmask function, 107, 108 grayshadow, 107 mouse movement, masking, 105 schematic with variable values, 108 shadow mask, 106, 107 z-index values, 106 closest-in limit, 96, 97 cursor, 109 distance and rounding values, 113, 114 events addListener, 111 bubble, 109 changebase function, 112 CHANGE button, 112 checkit function, 111 drawshadowmask function, 109 HTML coding, 112 init function, 109 mouseout event, 110 panning and zooming, 110 parallel structures, 112 pushcanvasunder function, 110 radio buttons, 112 showshadow function, 109 title indicating distance, 111 farthest-out view, 95, 96 Greenland problem, 95 latitude and longitude coordinate system, 98 distances between locations, 103 Drop LatLng marker option, 100, 103 equator at Greenwich prime meridian, 102 Greenwich prime meridian, 99 HTML5 application, 103 location, 91, 95 meridians, 99 parallels, 99 teardrop marker, 103 values, 99, 100 Wolfram Alpha, 101 opening screen, 89, 90 satellite view, 96, 97 semitransparent shadow, 91 shadow/spotlight, 90, 91 slider, zoom, 91, 92 zoomed in to limit, 97, 98 zooming out and moving north, 92, 93 Map portal, Google Maps, 129 API (see Google Maps API, map portal) application testing and uploading, 182 click not close to any target, 131, 132 content outline, 130 distances and tolerances, 144 external script file, 146, 147 hint button, 149

HTML5 markup and positioning, 147, 149
image-and-audio combination, 132 incorrect response, 135
Lego robot, 131
Liberty Island after panning and zooming in, 133
map centered on Dixon, Illinois, 136 mapmediabase application
functions, 160
latitude/longitude coordinates, 160 mapspotlight.html application, 160 portal code, 161
mapmediabase.html file, 130 mapmediaquiz.html file, 130 mapvideos application
canvas, 150 functions, 151 portal code, 151
mapvideos.html file, 130 mediacontent.js file, 130 media files, 130 mediaquizcontent.js file, 130
opening screen, 130, 131
panning west and zooming in Dixon, 134
piano music play, 135 project content, 141 project history and critical
requirements, 137
prompt concerning flute play, 136 quiz application, 134
code, 171, 172 external script file, 170 functions, 171
regular expressions, 145, 146 shuffling, 149, 150
video, audio, and image files, 129 video, audio and images presentation
and removal addEventListener, 142 checkit function, 143 display style, 143
div element, 142
last-viewed content removal, 143 media on demand, 142
style element, 142
video play, clicking Purchase College, 131
MasterCard numbers, 145 Math.floor method, 238 Math.min method, 60 mediacontent.js file, 146
Middleware. See PHP mountain function, 238

O

onChange attribute, 10

Open Source Miro Video Converter, 60 Origami directions, 225

application functions, 248 project code, 249

testing and uploading, 282 coordinate values, 236, 237 crease pattern, 228

critical requirements, 232, 233 first instructions, 226, 227

fish throat photograph, 230 fish with throat fixed, 231 kami, 226

line drawings/images, 225 mountain/valley folds, 228 opening screen, 226 origami definition, 225 origamifish.html, 226 paused video, sink step, 229 photograph display, 247 sink fold, 228

skinny vertical line, 227 step after sink, 229, 230 step line drawing functions

after making lips, 246

after wraparound steps, 246 built-in Math methods, 240

canvas coordinate transformations, 246

dividing a line into thirds and folding, 241

dividing-into-thirds step, 240, 241 HTML5 path-drawing facilities, 240 labeling at fold, half step, 245 labeling critical points, 243 littleguy function, 243, 244 rotatetfish function, 246, 247

sink center preparation, 245 triangle function, 242

triangleM function, 240, 242, 243 variables, 243, 244

steps array definition, 234

donext function, 234, 235 goback function, 234, 235 init function, 234 nextstep, 234

onLoad attribute, 234 origamifish.html, 233, 234

talking fish, 225, 226, 231, 232 unfolded fold line, 227

user interface, 235, 236 utility functions

calculation, 239, 240 display, 237–239

video presentation and removal, 247 origamifish.html application, 226, 282

P

Parallel structures, 112, 146 PHP, 383
character strings, 384 Form Action, 385, 387 function, 205 language, 384
script, 384, 385 SQL Queries, 387
recordset results, 387, 389 simple results, 387
three-tier model, 383 Pieceelements array, 295 piecesx value, 303 piecesy value, 303
Pixlr, 330, 332 playsink function, 247 playtalk function, 247 Point slope, 239
problemhandler function, 198 Proportion, 240
Pythagorean theorem, 237

R

Red-green-blue-alpha (rgba), 107 Relational databases, 377, 378 restorepreviousjigsaw function, 341, 342, 343

reversegeo function, 200

S

SATELLITE map, 140 Set typography, 232

setupgame function, 335 setupjigsaw function, 339 SQL, 379, 381

strokeStyle property, 7 strokeText method, 7 style.left value, 303 style.top value, 303

T

TERRAIN map, 138, 139 THIS element, 10 toFixed method, 114
touchHandler function, 302

U

US states game, 321 application
functions, 344 project code, 345
testing and uploading, 364 critical requirements, 329 doingjigsaw variable, 340 educational game, 321 elements creation, 335
Find the state, 322, 323 fullpage div, 342 image files
Adobe Flash symbol, 332 arrays, 335
base location, 332 bounding box, 332
Corel Paint Shop Pro, 330 Flash image export, 334 GIFs/PNGs, 335
Hawaii original symbol, 333 Hawaii with adjusted origin, 333,
334
illinoisclone, 332, 333
image-processing program, 335 imprecise positioning and sizing,
330
magic wand, 330, 331 offsets, 332
pixlr toolbar, 330 puzzle pieces, 329
selected Illinois state, 331 transparent backgrounds, 335
web-based pixlr image-editing tool, 330
jigsaw puzzle
correct arrangement, 328 feedback, 327, 328 pseudorandom processing, 326
Restore last jigsaw in process, 327 Save & close jigsaw, 327, 328 setting up, 339, 340
work in progress, 327 localStorage, 341
Name the state, 325 opening screen, 321, 322
response after correct answer, 325, 326 response to correct answer, 324 response to incorrect choice, 323
restore function, 339–341
Restore original/compress map, 325 restorepreviousjigsaw function, 341,
342, 343
spreading out pieces, 338, 339 Spread out states, 324
statesx and statesy arrays, 339 user interface

body element, 336 checkname function, 338 ev parameter, 337

HTML markup, 336 onsubmit attribute, 338 pickstate function, 337 setupfindstate function, 337

setupidentifystate function, 337 String method, 337

W

watchPosition method, 200 Web site database, 367

Wi-Fi hot spots, 194 Wi-Fi radios, 194 Wolfram Alpha, 101

World Wide Web Consortium, 2

X, Y, Z zIndex, 65, 106, 339.

[END](#)

For Part-2 :-

Chapter 7: Origami Directions: Using Math-Based Line Drawings Photographs, and Videos

- Introduction
- Critical Requirements
- HTML5, CSS, JavaScript Features, and Mathematics
- Overall Mechanism for Steps
- User Interface
- Coordinate Values
- Utility Functions for Display
- Utility Functions for Calculation
- Step Line Drawing Functions
- Displaying a Photograph
- Presenting and Removing a Video
- Building the Application and Making It Your Own
- Testing and Uploading the Application
- Summary

Chapter 8: Jigsaw Video: Using the Mouse and Touch to Arrange Images

- Introduction
- Background and Critical Requirements
- HTML5, CSS, JavaScript, and Programming Features
- Acquiring the Images and Data for the Pieces
- Dynamically Created Elements
- Setting Up the Jigsaw Puzzle
- Handling Mouse and Finger Touch Events
- Calculating If the Puzzle Is Complete
- Preparing, Positioning, and Playing the Video and Making It Hidden or Visible
- Building the Application and Making It Your Own
- Testing and Uploading the Application
- Summary

Chapter 9: US States Game: Building a Multiactivity Game

- Introduction
- Critical Requirements
- HTML5, CSS, JavaScript Features, Programming Techniques, and Image Processing
- Acquiring the Image Files for the Pieces and Determining Offsets
- Creating Elements Dynamically
- User Interface Overall
- User Interface for Asking the Player to Click a State
- User Interface for Asking the Player to Name a State
- Spreading Out the Pieces
- Setting Up the Jigsaw Puzzle
- Saving and Recreating the State of the Jigsaw Game and Restoring the Original Map
- Building the Application and Making It Your Own
- Testing and Uploading the Application
- Summary

Chapter 10: Web Site Database: Using PHP and MySQL

- Introduction
- Critical Requirements
- SQL, PHP, HTML5, and JavaScript Features
- Relational Databases
- SQL
- Local Storage
- Hash Function
- Client Side vs. Server Side for Input Validation
- Middleware: PHP
- Building the Application and Making It Your Own
- Testing and Uploading the Application
- Summary