2

PLOTTING MARKERS AND MESSAGE BOXES

Creating simple maps is a cool and useful way to see the area around a location, but you'll find creating maps even more fun and useful when you plot your own points on a map. Using mapping APIs you can overlay small graphics to call attention to locations (determined by latitude and longitude coordinates). Optionally, you can create messages that describe a location when the marker is clicked.

You've seen these principles in action on just about any chain store's website, among many others. If you're looking to shop in person, you've probably used the Find a Store link. From there, you enter your city, ZIP Code, address, or some other determination of your location. Then a map

© 2010 by Adam DuVander

appears showing the closest stores, with each store's location marked, often with a number that matches a results list. If the number is clickable, you will likely find that store's address, telephone number, or other information.

This chapter will get you started creating tools like store locators. You will learn to add markers, create custom icons, show messages in hovering boxes, and more. Mapping providers implement similar, but slightly different ways, of plotting markers on your map. Mapstraction wrangles these differences into a single set of functions that can add markers and message boxes no matter the underlying map type.

#1: Add a Marker to Your Map

The basic marker is a staple of web maps. Markers bring the user's attention to one or more points on the map. For many projects, you won't need to get any more complicated than a map and a handful of basic markers.

Although we'll be using Mapstraction to produce our markered maps, the underlying work is being done by whichever mapping service we're using. Just like the look of the map is determined by the provider, so will the default style of the basic marker. Figure 2-1 shows the differences among the markers from major map services.

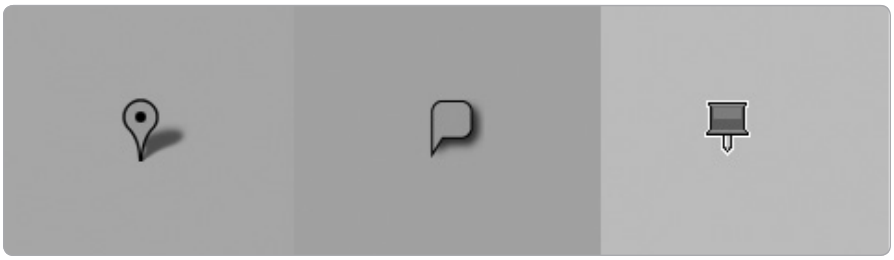

Figure 2-1: Default markers from different providers: Google, Yahoo!, and Microsoft

To add a simple marker to your map, you just need to use two Mapstraction functions. First, create the marker. Next, add it to the map. The reason for these two distinct steps will become clear in further projects when we start to use advanced options, such as custom marker icons.

Let's see what creating the marker looks like in code. Start with the basic Mapstraction map you created in "Create a Mapstraction Map" on page 10, and add these lines to the `create_map()` function:

```
marker = new mxn.Marker(new mxn.LatLonPoint(37.7740486,-122.4101883));
// marker options will go here
mapstraction.addMarker(marker);
```

The first line creates a marker object, passing latitude/longitude coordinates for the No Starch Press offices in San Francisco. Remember this is

Map Scripting 101
© 2010 by Adam DuVander

the same point we used as the center of our map in Chapter 1. By drawing attention to the graphical marker, we are essentially marking that spot as important.

The second line is a placeholder for any marker options we want to add later. (Any JavaScript line that begins with two slashes is a comment, and the browser ignores them.) The marker options are where we tell Mapstraction which icon to use or add a message to be displayed when the marker is clicked.

Finally, the third line adds the marker to the map. Once this happens, no additional options can be added. The reason is that the marker object is used only by Mapstraction. Once the marker is added to the map, however, Mapstraction makes the appropriate calls to the mapping provider. Mapstraction plots the marker based on all options set beforehand. In this case, we don't have options to add, but we'll add to this map in future projects.

If you're using Google as your mapping provider, your new map will look like Figure 2-2. The default Google icon sits in the center of the map. Although the marker is clickable, this marker is very simple and nothing actually happens if you click it. Read on to learn other cool things you can do with markers.

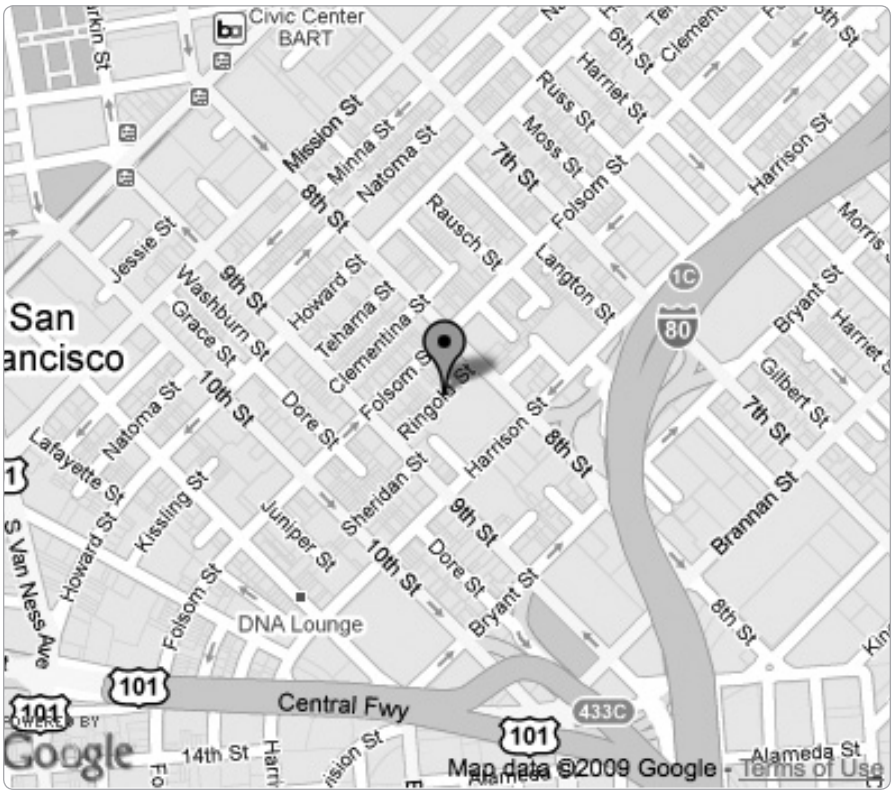

Figure 2-2: Google map with a simple marker

#2: Remove or Hide a Marker

Once your map has markers, you may wish to remove them from the map selectively. You might replace the current markers with new results. Or maybe the user added a filter that does not include the current marker. Mapstraction provides three functions to make markers disappear and reappear. Though *removing* and *hiding* may sound like similar terms, understanding the differences between them is important. Removing a marker from a map means the marker is gone for good. Simply hiding the marker allows you to make it visible again.

To use the functions to remove, hide, and show markers, you need access to Mapstraction and marker objects. These objects are generated when you create your new map and whenever you create a new marker. Whether they become available to the rest of your script, however, depends on the variable's scope.

Scope refers to the parts of code where a variable can be accessed. Any variable created inside the `create_map` function can only be used inside that function. To remove or hide markers, we need to make our Mapstraction and marker objects global. To do this, add this declaration right above the `create_map` function:

```
var mapstraction, marker;
```

These two variables now have a global scope, meaning we can use the variables outside of the `create_map` function. To remove a marker, you call the `removeMarker` function on the Mapstraction object:

```
mapstraction.removeMarker(marker);
```

To simply hide a marker, you call the `hide` function on the marker object:

```
marker.hide();
```

To make a hidden marker reappear, you call the `show` function on the marker object:

```
marker.show();
```

Where do you call these functions? Anywhere they're needed. For testing purposes, create a link anywhere after the `<body>` tag. For example, here is a link that will hide your marker:

```
<a href="javascript:marker.hide();">hide marker</a>
```

Again, this location is just for testing. You want unobtrusive JavaScript that isn't called from a link's `href`. One barrier to using all three of these functions is having access to the marker object.

Map Scripting 101
© 2010 by Adam DuVander

This single marker example only requires the variable be within a global scope. As you've seen, that's easy enough. When you start using many markers, you'll need a way to organize them beyond declaring dozens of variables.

Mapstraction's built-in ability to filter out certain markers (see "#9: Filter Out Certain Markers" on page 36) may be the easiest solution. If it does not provide all the features you need, you can always access all the markers that Mapstraction has added:

```
var allmarkers = mapstraction.markers;
```

Mapstraction's `markers` object gives you an array of markers. From here, you can remove, hide, or show them as you wish.

#3: Show a Message Box When Your Marker Is Clicked

Markers alone are useful because they identify spots on the map. Once your map has more than one, viewers will start wondering what each marker means. Sure, you could use custom icons to differentiate markers and we'll see how to do that shortly. But you can provide more information by showing descriptive text when the user clicks a marker.

Each mapping provider has a way to show a message box. Like markers themselves, the box looks different depending on the provider. Figure 2-3 shows the differences among the message boxes from the major map services.

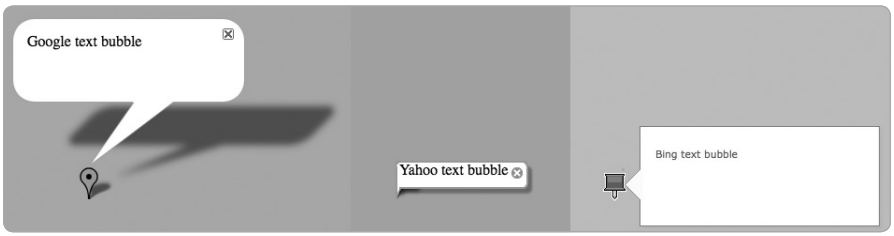

Figure 2-3: Message boxes from different providers

Mapstraction provides an interface, called an *InfoBubble*, that works with all providers. To create an InfoBubble for a marker, you add a marker option like so:

```
marker.setInfoBubble("Look ma, No Starch!");
```

The `setInfoBubble` function takes a string of text (HTML works, too) and saves it in connection with the marker. The line must be inserted after the marker object is created but before the marker is added to the map. If you have the code from creating a basic marker ("#1: Add a Marker to Your Map" on page 24), you can just add the `setInfoBubble` line in place of the comment about marker options.

© 2010 by Adam DuVander

For clarification, here are the commands necessary to create a brand new marker, include an InfoBubble, and place the marker on the map:

```
marker = new mxn.Marker(new mxn.LatLonPoint(37.7740486,-122.4101883));
marker.setInfoBubble("Look ma, No Starch!");
mapstraction.addMarker(marker);
```

Great! Now if you load this file, you see a basic marker in the No Starch Press neighborhood. Where is the InfoBubble? Click the marker, and you see something similar to Figure 2-4. Mapstraction and the mapping provider do all the work of capturing the click event and displaying the InfoBubble. All you need to do is provide the content. If you're hoping to open the InfoBubble automatically or from code, read on; I'll show you how to display a message box without making the user click in the next project.

Figure 2-4: Message box with message displayed

Map Scripting 101
© 2010 by Adam DuVander

#4: Show and Hide Message Boxes Without Clicking the Marker

Maps let users click around and interact with a location. You've seen how you can add clickable markers that provide more information about the spots you've plotted. But sometimes you want a little bit more control. Sometimes you want to open up that InfoBubble *without* the user's permission.

For example, if your map shows search results, you might duplicate the locations in a list format beside the map. Then users can choose a location from the list and its corresponding marker opens a message box on the map.

The basic setup for displaying a message box from code is the same as a standard clickable marker. You can just set some text as a marker option:

```
marker.setInfoBubble("Look ma, No Starch!");
```

This ensures that a clicked marker will still show your message. Then, from elsewhere in your code (such as when the user clicks one of the search results), you can tell the marker to open the InfoBubble:

```
marker.openBubble();
```

You can close the InfoBubble with a similar command:

```
marker.closeBubble();
```

The `openBubble` and `closeBubble` functions require the marker variable to be accessible globally. That is, the marker object needs to be declared at the top of your code, or you need to find another way to access it. In "Functions" on page 297, I describe variable scope and how to declare variables so they can be accessed anywhere in your code.

#5: Create a Custom Icon Marker

The quickest way to make a map feel like your own is to change the default icon used for markers. Mapstraction has simple marker options that make the technical process of using custom icons a cinch. The more laborious part may be creating the icon file itself. To avoid this, you can find icons others have made online for free. I list several resources at *http://mapscripting.com/download-custom-markers/*.

Still want to create your own? Read on.

Get Out the Image Editor

To create your own marker icon, you just need to have a graphics program that can save a transparent *.png* file. The icon can be whatever size you

Map Scripting 101
© 2010 by Adam DuVander

want, but keeping each dimension between 20 and 50 pixels is probably best. If the icon is too small, clicking it becomes difficult; too big, and the icon obscures the location you're attempting to call out.

If you're using Google as your mapping provider, you also want to create an image to use as your marker's shadow. This step isn't necessary if your marker is a similar shape to the Google default or if you're using another provider.

NOTE *Not much of an image magician? You can find an online service to create a shadow at* http://www.cycloloco.com/shadowmaker/.

Add Your Icon to the Map

Now that you have an icon, the easy part is adding it to the marker options. All it takes is setting a few values to tell Mapstraction where the icon image files resides. Your best bet is to keep custom marker icons in a special directory on your server. If you're testing locally, you can use local copies, accessed by their location relative to the page containing the map. For simplicity, I have the HTML file and the icon files in the same directory in this example. In reality, you might prefer to be more organized.

I decided to use a teensy No Starch Press logo for my custom icon. It's 27 pixels wide by 31 pixels high. Like I said, the icon is teensy. Then, I used a shadow-maker service to create a file that is 43×31 including the marker's shadow.

Finally, it's time to code. Add these lines as marker options. These lines are inserted after a marker has been created but before the marker has been added to the map:

```
marker.setIcon(❶'nostarch-logo.png', ❷[27,31]);
marker.setShadowIcon('nostarch-shadow.png', [43,31]);
```

The only parameter that you need to include is the path to the image ❶ for both the icon and the shadow. Notice that the dimensions of each graphic get passed as an inline array ❷. This parameter is optional but recommended. If you leave it out, some providers will assume the dimensions of the default marker, which could mean a poorly scaled graphic.

The results of the custom marker code are shown in Figure 2-5. The No Starch Press office is marked by the company's logo, a little iron icon. Notice the shadow, as well, which makes the graphic pop out from the map.

Omit the shadow icon at your own risk. Some mapping providers will assume the default shadow, which might look silly with your icon. Not every mapping provider uses shadows, but planning for one is good. If you really don't want a shadow, consider using a completely transparent graphic. I show an example of shadowless icons in "#69: Create a Weather Map" on page 237.

Map Scripting 101
© 2010 by Adam DuVander

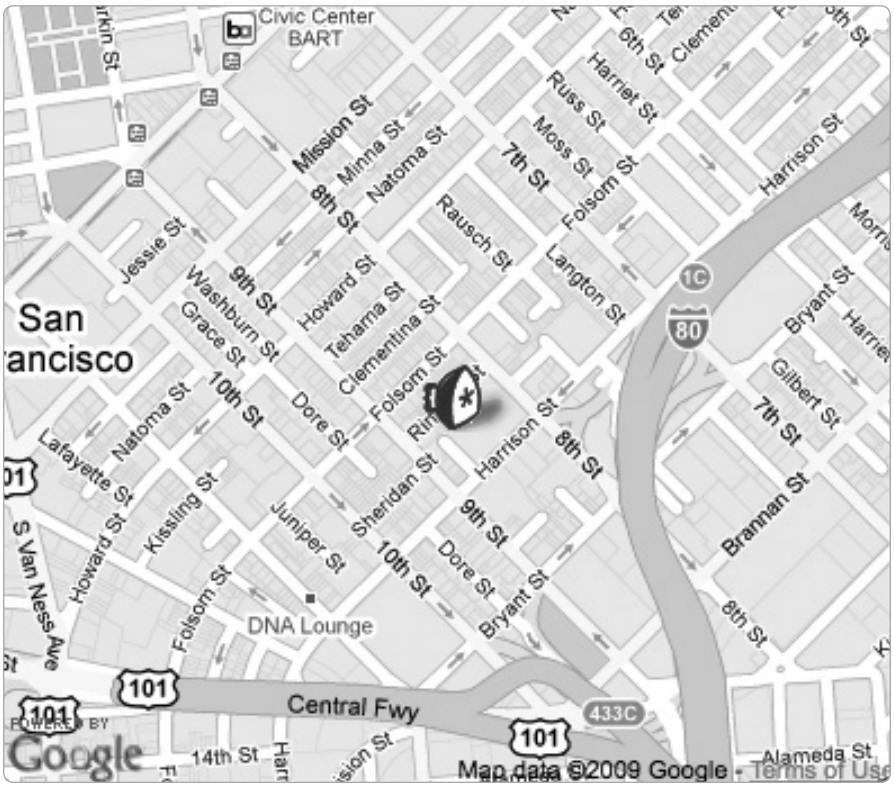

Figure 2-5: Custom marker shows the No Starch Press logo

#6: Create Numbered Markers

When you have a list of locations on your web page that you also want to plot on a map, provide users with numbered markers. For example, when displaying search results, you want a matching label both on and off the map so users can easily identify what's what.

Numbered markers are not any different from any other custom marker. You'll need to create a graphic icon for each number you want. Numerous icon sets are available online that you can use, or you can create them dynamically with the Google Charts API.

Generate the Numbered Icon

The Google Charts API generates reverse teardrop–style pins that look like the default Google marker. Using these Google-generated icons does not mean you have to use Google Maps. Mapstraction will add the icon for any provider to your map.

Map Scripting 101
© 2010 by Adam DuVander

You control the marker's background and border color, as well as what the label reads. The criteria you require are sent in the URL of the icon itself. For example, here is the URL for a red marker labeled with a number one:

```
http://chart.apis.google.com/chart?chst=d_map_pin_letter&chld=❶1|❷FF3333|❸000000
```

The final argument of the URL contains all the important information for the marker: the label text ❶ (in this case, the number one), the background color, ❷ and the border ❸ color. The colors are represented as hex values, similar to how colors are declared in CSS.

The individual pieces of the `chld` argument are separated by the pipe character, `|`. In a way, the final argument is really three arguments with its own way of segmenting the values.

Custom markers added to a map when using Google as a mapping provider also require a shadow. Because the shapes of these dynamic markers are all the same, the shadow can be static. The Google Charts API provides this URL:

```
http://chart.apis.google.com/chart?chst=d_map_pin_shadow
```

Now that you can generate the icons, you need to place them on the map. To do this, we'll call these Google Charts URLs on the fly.

Add the Icon to the Map

Armed with dynamically generated marker URLs from Google Charts, the process of adding these numbered markers to a map is much like adding any custom icon. Here is the code listing that creates five random points within San Francisco. Each marker is given an icon with a label numbered one through five based on the order that it is created:

```
mapstraction = new mxn.Mapstraction('mymap', 'googlev3');
mapstraction.setCenterAndZoom(new mxn.LatLonPoint(37.7740486,-122.4101883), 11);
mapstraction.addLargeControls();
for (i=1; i<=5; i++) {
    var rndlatlon = get_random_by_bounds(mapstraction.getBounds());
    marker = new mxn.Marker(rndlatlon);
    marker.setIcon(
      'http://chart.apis.google.com/chart?chst=d_map_pin_letter&chld=' + i +
      '|FF3333|000000', [21,32]);
    marker.setShadowIcon(
      'http://chart.apis.google.com/chart?chst=d_map_pin_shadow');
    mapstraction.addMarker(marker);
}
mapstraction.autoCenterAndZoom();
```


Map Scripting 101
© 2010 by Adam DuVander

The lines in bold set the generated icon and its shadow. The rest either sets up the map or creates the random points. For the code to work, you need a JavaScript function, `get_random_by_bounds`, which is discussed in Chapter 6 but which I have reprinted next. Put the previous code inside the `create_map` function used in all examples so far, and then make sure the following function is included somewhere in the JavaScript (but outside of other functions):

```
function get_random_by_bounds(bounds) {
  var lat = bounds.sw.lat + (Math.random() * (bounds.ne.lat - bounds.sw.lat));
  var lon = bounds.sw.lon + (Math.random() * (bounds.ne.lon - bounds.sw.lon));
  return new mxn.LatLonPoint(lat, lon);
}
```

Save your file. You'll see a map like the one shown in Figure 2-6 (marker locations vary—remember, they're random).

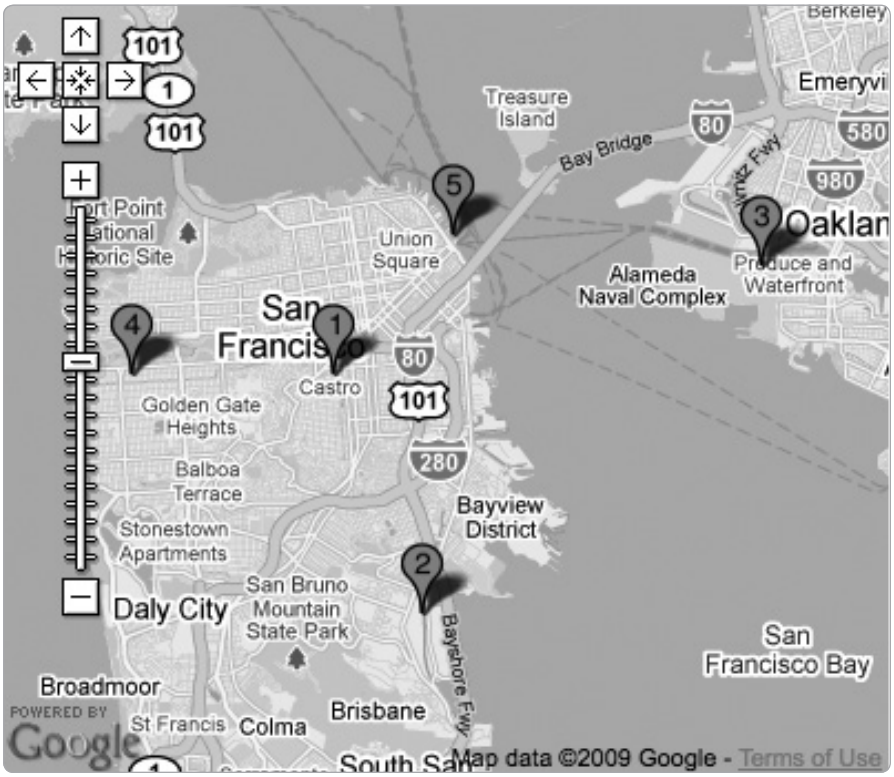

Figure 2-6: Numbered markers, randomly plotted

Use numbered markers when the order matters, such as when displaying nearby locations. Numbering is also helpful when users will match search results or another list from outside the map to the individual markers.

#7: Loop Through All Markers

When you've added a bunch of markers to the map, you may want a way to access them all. For example, you might be looking for outliers or determining which marker is the farthest north.

Mapstraction provides a property that holds an array of every marker plotted on the map. You can then reference an individual marker from within that array using standard JavaScript code to pull out a value at a specific index. Doing this for each marker on the map lets you loop through and perform an action on all the markers.

Add these lines to your code wherever you need to do something to each marker:

```
❶ var allm = mapstraction.markers;
❷ for (var i=0; i<allm.length; i++) {
❸   var thism = allm[i];
    // Any code for thism variable goes here
}
```

The first thing we do is reference the array of all markers from Mapstraction ❶ with a new variable name, `allm`. The saves us some typing, as we'll need to use the marker variable several times. Next, we use JavaScript's `for` statement ❷ to loop through the array. A temporary variable, `i`, keeps track of the index, as we count from zero (the first element in an array is at zero) up to the total number of markers.

As each marker becomes available, we place a reference to it inside the temporary `thism` variable ❸, a name I chose because it describes "this marker," as in the marker we are currently utilizing. Anything within the braces, { and }, of the `for` loop now has access to this new variable.

We can look up marker options or call functions on the marker (such as `showBubble` or `hide`, for example). In most cases, we cannot *add* options because options need to be added before the marker is added to the map. For example, we cannot change the marker's icon without removing and re-adding the marker.

Despite these few limitations, looping through the markers is a useful trick to add to your mapping tool bag. Many of Mapstraction's functions, such as filtering or autocentering, use a loop internally.

#8: Determine the Correct Zoom Level to Use Based on Markers

Once your map has several markers, ensuring that all of them can be viewed becomes a chore. This is especially true when your locations are being served up by a database ("#66: Plot Locations from a Database" on page 226). Markers start to fall outside of your manually set center and zoom levels.

Map Scripting 101
© 2010 by Adam DuVander

You may have tried to fix this on your own by changing the zoom level. If you zoom out, your markers can end up scrunched together, with lots of room to zoom in. The only way to achieve a good zoom level for any marker is to determine it programmatically after all the markers are added to the map.

Mapstraction makes setting the zoom level as easy as one function call. Add the following line to the `create_map` function from the basic map after you have added some markers:

```
mapstraction.autoCenterAndZoom();
```

You can also use a similar function that only works on displayed markers. That is, if you've hidden or filtered out some markers, you will probably want to zoom in to the ones that are still on the map. Instead of the previous function, use this:

```
mapstraction.visibleCenterAndZoom();
```

If these functions feel like magic, that's okay. Mapstraction makes it easy, but a lot is going on behind the curtain. Here's the run-down of how Mapstraction makes auto-zooming happen:

1. Loops through all the markers (or just the visible ones), and determines the maximum and minimum latitude and longitude of the markers. This measurement is called the *bounding box* and consists of four numbers that describe each edge of the box.
2. Finds the center of the bounding box by averaging the two latitudes and the two longitudes.
3. Checks zoom levels until it finds one that displays the entire bounding box.

Actually, Mapstraction does not need to perform the last two steps for very many mapping providers. Most already have something that does the auto-zooming work within their own library. That wasn't always the case, however, and it points to the power of Mapstraction. Mapstraction is able to add these indispensable functions before they're available in all map APIs.

To get a feel for how auto-zooming works, insert this code in the basic map's `create_map` function to add random markers:

```
❶ var num_markers = 5;
❷ var bigbounds = new mxn.BoundingBox(37.766, -122.400, 37.784, -122.418);
  for (i=1; i<=num_markers; i++) {
      var rndlatlon = ❸get_random_by_bounds(bigbounds);
      marker = new mxn.Marker(rndlatlon);
      mapstraction.addMarker(marker);
  }
❹ mapstraction.autoCenterAndZoom();
```

This code chooses five random markers, but you can change the first variable ❶ to any number you want. I've also created a bounding box ❷

© 2010 by Adam DuVander

that is larger than the basic map's visible area (you can learn more about bounds in Chapter 6). These bounds are used to produce a new random point for each of my markers. You actually need to include a special function ❸ to create the point. We'll get to that in a moment.

First, note the last line ❹, the one that auto-zooms. Try commenting it out by placing a // at the front of the line to see how the markers look without auto-zooming. Reload the map a few times with and without the comment slashes. Figure 2-7 shows an example comparison of the maps in each of these situations.

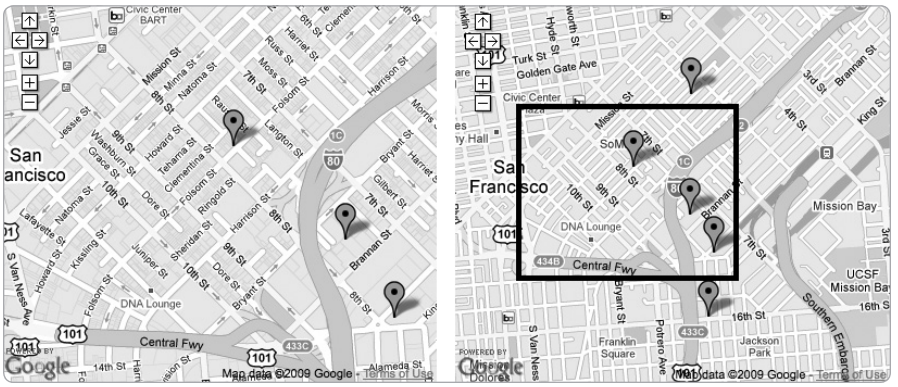

Figure 2-7: Difference between markers without and with automatic centering and zooming

Of course, you need that special function, `get_random_by_bounds`. Unlike in the previous code, add this outside of the `create_map` function but within the JavaScript section:

```
function get_random_by_bounds(bounds) {
  var lat = bounds.sw.lat + (Math.random() * (bounds.ne.lat - bounds.sw.lat));
  var lon = bounds.sw.lon + (Math.random() * (bounds.ne.lon - bounds.sw.lon));
  return new mxn.LatLonPoint(lat, lon);
}
```

This function is described in detail in "#44: Get a Random Point in a Bounding Box" on page 140.

As for the automatic centering and zooming, now that you can do it in a single function call, you'll likely include it for all but the simplest maps—it's really that useful.

#9: Filter Out Certain Markers

You must have a map with a whole bunch of markers by now, right? That means you're getting the hang of this mapping stuff. With a full screen of markers, users will likely want to way to see only what they care about. That's where you'll find Mapstraction's filtering options handy.

Map Scripting 101
© 2010 by Adam DuVander

Keeping track of many markers without the filtering options is a pain. You need to maintain global arrays or make use of the `mapstraction.markers` object. In either case, you need a way to distinguish the type of marker or some data associated with it.

The first step in filtering is to create a new attribute and add it to your new marker. You do this by adding some marker options, which has to happen after a marker is created but before you add it to the map. Here, I'll set the price to be 1000—maybe the marker represents an apartment and this attribute is its rent:

```
marker.setAttribute('price', '1000');
```

If you'd like to add more attributes, such as number of bedrooms, you can do that with additional `setAttribute` lines. Once you've added the data for several markers, you can move on to filtering.

Filters are applied after the markers have been added to the map. In fact, filtering usually happens in response to user behavior, such as when a user clicks a filter button or enters search terms.

To show only markers with a price attribute greater than or equal to 1000, use this code:

```
mapstraction.removeAllFilters();
mapstraction.addFilter(❶'price', ❷'ge', ❸1000);
❹ mapstraction.doFilter();
```

First, use `removeAllFilters` unless you know no filters are applied. The reason is that filters are additive, meaning a second filter does not remove the first. You could end up with fewer results than you expect because of a previously applied filter.

Once filters are removed, you can continue. To add the filter requires three parameters: the attribute name ❶, the operator (in this case **g**reater than or **e**qual to) ❷, and the number ❸ to use as a comparison. Finally, the map will not change at all unless you apply the filter ❹.

Table 2-1: Filtering Operators

Operator	Description
`ge`	Greater than or equal to—use on numbers.
`le`	Less than or equal to—use on numbers.
`eq`	Equal to—use on numbers or with words (such as tags or types).

Once a filter is applied, the markers that *don't* match the filter will disappear. In our example, anything with a price attribute less than 1000 (or without a price attribute) will be removed. Thus, filtering can be thought of as filtering *in* rather than *out*.

Mapstraction provides three operators to use to filter markers, as seen in Table 2-1. You can combine filters to achieve more granular results.

© 2010 by Adam DuVander

Sticking with the apartment search theme, you might add a neighborhood attribute, so users can view apartments in a certain neighborhood that are below a certain price, for example.

Mapstraction filters are a speedy way to show only a subset of markers based on simple criteria. They do not require any additional communication with the server because Mapstraction stores information about every marker in memory. For an example of filtering used in a real project, see "#71: Search Music Events by Location" on page 260.

#10: Remove or Hide All Markers

Need to start fresh, or want to show a clear map in some situations? We all can use a little spring cleaning from time to time. I've already shown how to remove or hide a single marker in this chapter. Now we'll get rid of them all.

Again, make sure you understand that when you remove a marker it is gone forever. Removing a marker is sometimes desired, such as when the user activates a new search. Mapstraction has a function to achieve a clean slate. A hidden marker, on the other hand, can always be shown again. We'll have to write our own function to hide all markers and make the slate *appear* clean.

First, let's be destructive and remove all the markers from our map. Add this line wherever you want to axe all the markers:

```
mapstraction.removeAllMarkers();
```

That's it—they're gone. "#71: Search Music Events by Location" on page 260 shows an example of this function in use every time a user starts a search. By removing the markers, we ensure the previous search results don't get mixed up with the new ones.

If we only want to hide all markers, we need to write our own function. To do this, we need to loop through all markers (described in detail in "#7: Loop Through All Markers" on page 34) and hide each one.

```
function hideAllMarkers() {
❶  var allm = mapstraction.markers;
   for (var i=0; i<allm.length; i++) {
❷    var thism = allm[i];
❸    thism.hide();
   }
}
```

So far, much of our code has been used inside the `create_map` function. Because `hideAllMarkers` is a new function, we need to add it in its own place outside of other functions but still in the JavaScript section of the page.

The function itself is straightforward. It first grabs a reference to the marker object ❶ from Mapstraction, which holds an array of every marker added to the map. Then, using that array of markers, the function goes

Map Scripting 101
© 2010 by Adam DuVander

through them one by one. Each time through the loop, the function takes another marker and puts it in a temporary variable named `thism` ❷ (for "this marker"). Finally, it calls the `hide` function ❸ on this marker.

By the end of the function, no markers will be displayed on the map. We've looped through every marker and hidden them one at a time.

To write the function is not enough. We need to call the function from somewhere else in our code:

```
hideAllMarkers();
```

Notice that this call looks very similar to the call for removing all the markers. One major difference is that `removeAllMarkers` was called on the Mapstraction object. This new function is merely called on its own. The difference is that we wrote `hideAllMarkers` in our own code, whereas Mapstraction's functions are part if its package.

Writing utility functions, as we did here to hide every marker, is an important part of programming. Now that we've written the function once, we can call it any time we need it.

#11: Handle Clusters of Markers

This chapter has already covered several ways to make sense of a map with many markers plotted. You can number them and filter them. You can automatically zoom to show all the markers within the visible portion of the map. These tools are all good to have, but you'll find that sometimes your markers are still scrunched together and overlapping. You can't avoid it, but you can make it less of a problem.

Instead of showing every single marker, you can use a special icon to represent a cluster of markers. Then, when users zoom in, the cluster will disappear and be replaced by the actual markers. You can see an example of many markers, with and without clustering, in Figure 2-8.

Figure 2-8: Difference between markers with and without clustering

The code behind marker clustering is surprisingly complicated, but the concept is simple. Although many approaches exist, commonly the map is divided into a grid. If one *cell* of the map contains more than one marker (or more than a certain number—you may prefer a cutoff of five markers per cell), they're replaced by a cluster.

Rather than write this algorithm ourselves, we'll use a utility that is already written to work directly with Google Maps, called *ClusterMarker*. You can download the code from *http://www.acme.com/javascript/* and save it in a file named *clusterer2.js*.

Unlike most examples in this book, you'll work with Google Maps directly, as you did in "Create a Google Map" on page 7. In addition to including the Google Maps API JavaScript in the header, you also need to reference the new cluster file. Add this to the header section of your HTML:

```
<script type="text/javascript" src="clusterer2.js"></script>
```

The cluster code is similar to Mapstraction in that it wraps itself around Google Maps. The code to add markers will go through the cluster functions and then be routed to Google Maps. Replace your `create_map` function with this one, which will place 100 random markers on the map and cluster where necessary:

```
function create_map() {
  if (GBrowserIsCompatible()) {
    // Basic Google Map
    var map = new Gmap2(document.getElementById("mymap"));
    var center_point = new GLatLng(39.34, -98.26);
    map.setCenter(center_point, 8);
    map.addControl(new GsmallMapControl());
    // Cluster settings
❶    var clustobj = new Clusterer(map);
❷     clustobj.SetMaxVisibleMarkers(50);
❸     clustobj.SetMinMarkersPerCluster(2);
    // Add Markers
    for (var i=1; i<=100; i++) {
      var lat = center_point.lat() + Math.random() - 0.5;
      var lon = center_point.lng() + Math.random() - 0.5;
      var gmk = new GMarker(new GLatLng(lat, lon));
❹     clustobj.AddMarker(gmk, 'Marker #' + i);
    }
}
```

I've centered this map roughly in the middle of the United States (hello, Kansas!). Once the basic map has been created, we need to tell the cluster code where to find it ❶, which creates an object that is put into a variable named `clustobj`.

Map Scripting 101
© 2010 by Adam DuVander

Before we add any markers, we want to reset some properties of the clusterer. The first ❷ sets the number of markers when the cluster code will begin clustering. The default is 150, which means every single one of our 100 markers will be shown without clustering if we don't change this setting. The next setting ❸ declares how many markers need to occupy a grid cell before clustering takes over. The default is 5 and, for our example, seems a little too crowded. Experiment with what works best for your map.

Now we're ready to add markers to the map. I wrote a `for` loop that creates 100 markers at random points near the center of our map. Then, instead of adding them directly to the map, we add them to the clusterer ❹.

Save your file, and load it in a browser to see the large clustered markers. You may also see a few stray normal markers—these markers didn't need to be clustered because other markers weren't nearby. The map looks a whole lot cleaner, doesn't it? Zoom in and some of the clusters will disappear, as the map is able to display the actual markers without crowding them.

Change the Cluster Icon

Out of the box, your cluster code uses a large, blue icon for cluster markers. If you have another graphic you would rather use, you can include it instead.

Add the following code after your other cluster settings but before you start adding markers:

```
var cicon = new GIcon();
cicon.image = 'icon.png';
cicon.iconSize = new GSize(27,31);
cicon.shadow = 'shadow.png';
cicon.shadowSize = new Gsize(43,31);
clusterer.SetIcon(cicon);
```

Clustering icons solves the marker overload problem, in which the markers are so numerous they become meaningless. Clustering is also a quick way to still show everything without overwhelming your users.

© 2010 by Adam DuVander