PART 4

Exploring Other Mashup Topics

Now that you've had some experience of the anatomy of a mashup and come to grips with some specifics, this part will apply what you've learned to a range of different APIs and technologies. We'll examine a different category in each chapter, starting with online maps in Chapter 13 and moving onto social bookmarking, online calendars, online storage, office documents, microformats, and searches.

CHAPTER 13

Remixing Online Maps and 3D Digital Globes

It would be difficult to overstate the importance of maps in the course of human civilization. Maps help us place ourselves in a spatial context with regard to everything else on the planet. With the advent of the Web, we have been able to access online maps, which have proven to be both useful and fascinating. There are many practical daily uses for these maps, including getting driving directions, locating a restaurant in the neighborhood, thinking of places to go for travel. Online maps let us explore parts of the world in new ways too. Moreover, maps provide an intuitive conceptual and visual metaphor/space for connecting other things; as part of our cultural development, we have all developed a strong intuition for maps.

It is no wonder then that online maps have been used extensively in many mashups. One reason for this extensive activity is that contemporary online maps are designed for easy customization. In this chapter, you will learn how to customize maps. It is an exciting time for web-based mapping, and we really are only at the beginning of developing this immersive space. Add the Global Positioning System (GPS), more immersive systems/platforms such as Google Earth and Second Life, ubiquitous computing, and GPS devices, and we're going to get amazing stuff.

Note We're good at reading maps, and we know how things on maps are related to each other. We're used to adding dots and drawing lines. Hence, it's not much of a stretch for us to add other things—dots, lines, pictures, and even more abstract data to maps. Things that are located in space have a natural spot on maps. That's what I mean by saying that maps are a powerful metaphor.

The goal of this chapter is to introduce how to use some leading systems for remix purposes (Google Maps, Yahoo! Maps, Microsoft Maps, MapQuest, and Google Earth), looking for commonalities and differences. A potential framing question, technically, is how to write a wrapper so that one can substitute one system for another—and I will tell you about a couple of such efforts. However, most people want to use just one of these maps and do some easy customization; hence, I will show how you can do that. Each system has strengths, and it's useful to be able to interchange information among them without much effort. Obviously, I will not attempt to exhaust this very rich subject, but I'll provide you with a strong starting point to build on.

In this chapter, I will cover the following:

* I'll describe how to use the APIs of the major map providers, such as Google Maps, Yahoo! Maps, and Microsoft's Live Search Maps.
* I'll describe how you can make web-based maps without programming.
* I'll describe declarative approaches to working with maps, such as creating KML and GeoRSS and CSV.
* I'll teach you the fundamentals of KML and how to do some basic programming of Google Earth.
* I'll show you how to create a mashup of Flickr, Google Earth, and Google Maps using KML.

To learn more about the subject of online maps, please read the following:

* *Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional* by Michael Purvis, Jeffrey Sambells, and Cameron Turner (Apress, 2006)
* *Beginning Google Maps Applications with Rails and Ajax: From Novice to Professional* by Andre Lewis, Michael Purvis, Jeffrey Sambells, and Cameron Turner (Apress, 2007)

The Number of Online Maps

The capability of individual users to make web-based digital maps has been rapidly increasing over the past several years. Online maps have evolved quickly from maps with only predefined purposes (for example, driving directions) to increasingly customizable platforms. That is, we are close to having map-making for the masses—Geographic Information System (GIS) for dummies (so to speak).

Perhaps the most dramatic revelation of the capabilities of what would later be known as Ajax was the emergence of Google Maps in February 2005.[1] It was a watershed event for all web apps and showed that it was possible to have highly interactive apps on a large scale. (Yes, people had been using JavaScript for menus but not for shipping a substantial amount of real-time data.) In the area of maps, this event marked the beginning of what I will refer to as *new-style* online maps as opposed to old-style online maps (which are an endangered species it would seem; even MapQuest has switched over to Ajax-type maps). By old-style online maps, I mean non-JavaScript-powered maps—ones in which moving around or zooming means reloading the page.

The most obvious aspect of the new-style maps is the substantial increase in interactivity (with the fluid drag-and-drop capabilities, instead of clicking and waiting for a page reload). However, hackers quickly realized that the Ajax technology also allowed Google Maps to be extended to new purposes.[2] Apps that showed up included Housingmaps.com. These apps, however, involved the extensive reverse engineering of Google Maps; the techniques that emerged could break anytime, and the whole enterprise was of questionable legality and longevity. What Google did was then smart and novel: it released an API to formalize and regulate the usage of its maps, transforming Google Maps mashups into legitimate business. Google really did transform the whole endeavor of GIS through Google Maps by making online maps accessible to and customizable by the masses. Competitors soon followed. Yahoo!, Microsoft, and eventually MapQuest all went new-style, with the release of not only Ajax implementations but APIs to boot.

1. http://en.wikipedia.org/wiki/Google_maps and http://www.adaptivepath.com/publications/essays/archives/000385.php

2. http://www.oreillynet.com/etel/blog/2005/05/hackers_tap_into_the_functiona.html

Examples of Map-Based Mashups

Before we figure out how to make map-based mashups, it's handy to look at a number of examples to understand what is possible. As discussed in Chapter 1, there are many map-based mashups, including Housingmaps.com, Chicagocrime.org, and the 1,000+ map-based mashups listed on Programmableweb.com (`http://www.programmableweb.com/tag/mapping`). Here are some specific examples:

* `http://tutorlinker.com/` connects tutors to students via a mapping interface.
* `http://flashearth.com` is a mashup of various major online map services displayed through a Flash interface.

Programmableweb.com lists Google Maps as by far the most popular API used in mashups. Mapping as a category is very popular. Yahoo! Maps and Microsoft's Live Search Maps are also in the top ten.

Making Maps Without Programming

In this chapter, I'll show you how to create online maps without any programming before we jump into programming online maps. Then I'll show you how to use Mapbuilder.net as an example of a third-party authoring tool for Google Maps before discussing the functionality available directly in Google Maps and Virtual Earth for making custom maps. Along the way, I'll explain how you can transform a collection from Yahoo! Local into CSV format that you can send to Mapbuilder.net.

Mapbuilder.net

The scenario I'll focus on in this chapter is building a map with dots pointing to a list of places for which you have addresses in the United States. We will look at more sophisticated scenarios later.

Let's see how much of a Google map you can build without any programming. For the first two years after Google Maps was released, there was no built-in user functionality to create custom maps. You could use the Google Maps API to create custom maps, but that's beyond the skill or energy level of most users. Andriy Bidochko built Mapbuilder.net, a free service to let users create Google and Yahoo! Maps with custom markers through an interface that does not require knowledge of JavaScript. In this section, I'll show you how to use Mapbuilder.net as an easy way to create your own map.

In April 2007, Google introduced the My Maps feature into its online maps. Moreover, Microsoft's Live Search Maps has similar collection-making functionality in its maps. That the big players have incorporated functionality that allows its end users to create maps beyond locating a single location or displaying driving directions between two points is validation of the map builder concept. I also describe how to use those functions.

What are some sites built using Mapbuilder.net? Drawing from the list of featured maps,[3] I see ones like Edinburgh Pub Guide.[4] Note some have been heavily used in commercial contexts: the most popular of all time for Mapbuilder.net-built maps[5] is the "Find a Distributor" map for Pacific Wireless.[6]

Mapbuilder.net How-To

The following are the step-by-step instructions for how to create a map using Mapbuilder.net:

1. Sign up for an account.[7]
2. Click the New Map link.[8] (Note that the map name must contain only letters, numbers, underscores, and dashes.)
3. You can create a dot in one of two ways:
    a. Start typing addresses to add (under Location Search & Quick Navigation).
    b. Click the map to indicate a spot's location.
4. When you add a new dot, it will be flashing. Click any marker on the map to update its information or delete it (using the Update or Delete button). Remember to hit the Add button to save the location.
5. Use the Save Center, Zoom, MapType button located on the map to save the center of your map, zoom level, and map type (regular map, satellite, hybrid) as well. (You will have to adjust the scale and center of the map by using the zoom control and dragging the mouse to fit your taste; the default is not that helpful.)
6. Click the Preview link (located at the upper right) to take a look at the map.

7. To embed the map on your own site as a Google map, click the Source Code link, and copy the displayed code to your own site. (You will need to enter the appropriate API key in the code. See the "Google Maps API" section later in this chapter for how to get a key.) Note that you can use the options listed under Map Controls and Map Implementation to set other options for the map, including getting access to code to create a Yahoo! Map version of your map.

With a bit of futzing and by doing a series of searches on Yahoo! Local for addresses, I created "Some of my favorite bookstores around Berkeley,"[9] as hosted on Mapbuilder.net. I also embedded the map elsewhere using both Yahoo! Maps and Google Maps:

* "Some bookstores I like" (Google version),[10] kept up-to-date via JavaScript injection[11]
* "Some bookstores I like" (Yahoo! version)[12]

Overall, I recommend Mapbuilder.net as a way to quickly build custom Google maps or Yahoo! maps or as a way to get started learning the APIs.[13]

3. http://www.mapbuilder.net/About.php
4. http://www.imkblue.pwp.blueyonder.co.uk/index.html
5. http://www.mapbuilder.net/Popular.php?OP=ALL
6. http://www.pacwireless.com/distributor-locations.shtml
7. http://www.mapbuilder.net/SignUp.php
8. http://www.mapbuilder.net/Map.Add.php
9. http://www.mapbuilder.net/users/rdhyee/9329

Google My Maps

In addition to Mapbuilder.net, consider using Google's built-in My Maps functionality to create a custom Google map, which is documented here:

```
http://local.google.com/support/bin/answer.py?hl=en&answer=68480
```

I created a Google map with the same three bookstores as the previous example. I could either show the map hosted under the Google domain here:

```
http://maps.google.com/maps/ms?f=q&hl=en&geocode=&ie=UTF8&msa=0~CCC
&msid=116029721704976049577.0000011345e68993fc0e7&z=14&om=1
```

or embed it elsewhere by copying and pasting HTML for an embeddable iframe to generate the following, for example:[14]

```
http://examples.mashupguide.net/ch13/embedded.GMap.html
```

There are some major advantages of using My Maps. It's well integrated into Google Maps with its Search the Map functionality for looking up addresses and the Find Businesses option for locating businesses by name. If a marker comes up in your search results, you can click the Save to My Maps link to save the location on one of your custom maps. In addition, My Maps also lets you edit the markers and draw lines and polygons on your maps. Altogether, Google My Maps goes a long way to letting end users create custom maps (based on Google Maps, of course) without knowing JavaScript.

From a mashup point of view, you should know that you can generate a KML version of one of any of the maps produced by My Maps. The `msid` parameter of the Google Maps URL holds an identifier for a My Map–produced map (for example, `116029721704976049577.0000011345e68993fc0e7` for the map I produced). A URL of this format:

```
http://maps.google.com/maps/ms?f=q&msa=0&output=kml&msid={my-map-id}
```

such as the following:

```
http://maps.google.com/maps/ms?f=q&msa=0&output=kml~CCC
&msid=116029721704976049577.0000011345e68993fc0e7
```

returns a KML version of the map. As I describe in greater detail later in this chapter, KML (an XML vocabulary for displaying geospatial information) lets you easily parse details of the map.

10. http://examples.mashupguide.net/ch13/SomeBookstoresGMap.html

11. For an explanation of how to use JavaScript injection, see http://www.mapbuilder.net/Map.

Implemenation.php.

12. http://examples.mashupguide.net/ch13/SomeBookstoresYMap.html

13. I ran into one snag: I wasn't able to delete a certain point using Firefox 1.5.0.7 on Windows XP no matter what I did. I finally deleted the point by logging in to Mapbuilder.net using Opera.

14. http://local.google.com/support/bin/answer.py?answer=72644

A Mashup Opportunity: Mapping Yahoo! Local Collections

Yahoo! Local makes it easy to make collections of places, but it does not allow easy mapping of those places. For instance, I assembled a collection of bookstores around Berkeley,[15] but the Yahoo! interface does not allow me to easily map those bookstores. In this section, we will create this by transforming one data format to another (specifically, transforming the addresses of stores selling used books offered in the XML that comes from the Yahoo! Local API into CSV, which is understood by Mapbuilder.net). There are two steps to this, which I will cover in more detail in the following sections:

1. Get information out via the Yahoo! Local API.
2. Transform the collection appropriately (XML to CSV) to feed into Mapbuilder.net.

Getting XML Out of Yahoo! Local via getCollection

We will use the `getCollection` method of the Yahoo! Local web service, which "enables you to get detailed information about a collection created with Yahoo! Local collections, through a REST-like API."[16] So, how do we use it?

The base URL is as follows:

```
http://collections.local.yahooapis.com/LocalSearchService/V1/getCollection
```

There are four parameters, as shown in Table 13-1.

Table 13-1. getCollection Parameters

Parameter	Description of Parameter	Value of the Parameter
appid	Application ID	raymondyee.net
collection_id	ID of the collection to be retrieved	1000014156
output	Output type	Unspecified, thus returning the default of XML
callback	Callback function	Unspecified

In other words, we can formulate the following query:

```
http://collections.local.yahooapis.com/LocalSearchService/V1/getCollection?~CC
C
appid={app-id}&collection_id=1000014156
```

Tip You can get your Yahoo! addid at https://developer.yahoo.com/wsregapp/index.php.

15. http://local.yahoo.com/collections?cid=1000014156

16. http://developer.yahoo.com/local/V1/getCollection.html

This request returns the code shown in Listing 13-1.

Listing 13-1. XML for a Collection from Yahoo! Local

```
<?xml version="1.0" encoding="UTF-8" ?>
<Result id="1000014156" xmlns="urn:yahoo:travel"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="unknown">
  <Title>bookstores around Berkeley</Title>
  <Description>some of my favorite bookstores around Berkeley.</Description>
  <CreatedTime>2006-10-24 13:29:40</CreatedTime>
  <Username>Raymond Yee</Username>
  <CommentCount>0</CommentCount>
  <Item>
    <Address>
      <Address1>2476 Telegraph Ave</Address1>
      <Address2 />
      <City>Berkeley</City>
      <State>CA</State>
      <PostalCode>94704</PostalCode>
    </Address>
    <id>21518795</id>
    <Title>Moe&apos;s Books</Title>
    <CreatedTime>2006-10-24 13:29:41</CreatedTime>
    <Description>
      * The Bay Area&apos;s Largest Selection of Used Scholarly Books
    </Description>
    <Url>
http://local.yahoo.com/details?id=21518795&amp;stx=&amp;csz=Berkeley+CA&amp;
```

```
ed=xoDOxa160SyYoswS6OvDhQk64pj4Q8RHG5PQhcSqprzxVT6mDHMezwfQ2U244pugG4LDSdibA78iS
w--
    </Url>
    <type>Retail Shopping</type>
    <Category>Used &amp; Rare Bookstores</Category>
    <Photo />
    <Tag />
    <Phone>(510) 849-2087</Phone>
  </Item>
  <Item>
    <Address>
      <Address1>1730 4th St</Address1>
      <Address2 />
      <City>Berkeley</City>
      <State>CA</State>
      <PostalCode>94710</PostalCode>
    </Address>
    <id>21512172</id>
    <Title>Cody&apos;s Books</Title>
    <CreatedTime>2006-10-24 14:07:28</CreatedTime>
    <Description />
    <Url>
http://local.yahoo.com/details?id=21512172&amp;stx=&amp;csz=Berkeley+CA&amp;
ed=3uqWba160SzFEqntYzu46yunejqBEmJnCBEi_I7QbD68sZTEVRYl4WkOGEf6alVIaEB3</Url>
    <type>Retail Shopping</type>
    <Category>Bookstores</Category>
    <Photo />
    <Tag />
    <Phone>(510) 559-9500</Phone>
  </Item>
  <Item>
    <Address>
      <Address1>6060 El Cerrito Plz</Address1>
      <Address2 />
      <City>El Cerrito</City>
      <State>CA</State>
      <PostalCode>94530</PostalCode>
    </Address>
    <id>21414999</id>
    <Title>Barnes &amp; Noble Booksellers</Title>
    <CreatedTime>2006-10-24 14:07:56</CreatedTime>
    <Description />
    <Url>
http://local.yahoo.com/details?id=21414999&amp;stx=&amp;csz=El+Cerrito+CA&amp;
ed=Fo51gq160Sy24fPx_u7IvyZen3kxQq5wR9ZOi_Aos2J.pPlJ75D_th3K2MHtNCWF_V5k_nOq62ssy
3I-
    </Url>
    <type>Retail Shopping</type>
    <Category>Bookstores</Category>
    <Photo />
    <Tag />
    <Phone>(510) 524-0087</Phone>
  </Item>
</Result>
```

Transforming the Yahoo! Local XML into CSV for Mapbuilder.net

Our goal is to convert the XML data from Listing 13-1 into CSV. There are various techniques you could consider to do this. One way is to write a PHP script that takes a collection ID and outputs CSV, as shown in Listing 13-2.[17]

17. http://examples.mashupguide.net/ch13/yahooCollectionToCSV.php

Listing 13-2. PHP Script to Convert Yahoo! Local XML to CSV

```
<?php

function getResource($url){
  $chandle = curl_init();
  curl_setopt($chandle, CURLOPT_URL, $url);
  curl_setopt($chandle, CURLOPT_RETURNTRANSFER, 1);
  $result = curl_exec($chandle);
  curl_close($chandle);
  return $result;
}

// get a collection_id
//default to my own
  $cid  = isset($_REQUEST['cid']) ? $_REQUEST['cid'] : "1000014156";

  $url = ~CCC

"http://collections.local.yahooapis.com/LocalSearchService/V1/getCollection?~C
CC
appid=[app-id]&collection_id=". urlencode($cid);
  $feed = getResource($url);
  $xml = simplexml_load_string($feed);

  //header("Content-Type:text/csv");
  $out = fopen('php://output', 'w');

  $header = array("Caption","Street Address","City","State","Zip");
  fputcsv($out, $header);

   foreach ($xml->Item as $item) {
    $caption = $item->Title;
    $street_address = $item->Address->Address1;
    $city = $item->Address->City;
    $state = $item->Address->State;
    $zip = $item->Address->PostalCode;
    fputcsv($out, array($caption,$street_address,$city,$state,$zip));
  }

    fclose($out);
?>
```

With this code in hand, we can generate a CSV file that we can feed to Mapbuilder.net:

```
http://examples.mashupguide.net/ch13/yahooCollectionToCSV.php?cid=1000014156
```

To try this, we can go to our collections here:

```
http://local.yahoo.com/userreviews?target=pOTJ1rUjf64lpQPwpZGZmXVTOyaM~CCC
&rvwtype=COLLECTION
```

and pull out collection ID numbers to feed to the script to generate the CSV:

```
Caption,"Street Address",City,State,Zip
"Moe's Books","2476 Telegraph Ave",Berkeley,CA,94704
"Cody's Books","1730 4th St",Berkeley,CA,94710
"Barnes & Noble Booksellers","6060 El Cerrito Plz","El Cerrito",CA,94530
```

We can then take the CSV and feed it to Mapbuilder.net to create our map.

Note You might say that writing such a script gets you only CSV and you still have to manually create a map with Mapbuilder.net—and you're right. Later, we'll transform this collection into formats that are easier to work with in terms of generating maps.

Collection Building in Microsoft's Live Search Maps

For Microsoft's Live Search Maps (`http://local.live.com`), powered by Microsoft's Virtual Earth, the story is a bit more complicated. Live Search Maps has excellent collection-building facilities. I was surprised how easy it was to look up bookstores and save them to collections and then to see those bookstores on the map all within the Microsoft map environment. If you make a collection public, anyone can access it here:

```
http://maps.live.com/?v=2&cid={collection-id}&encType=1
```

For example:

```
http://maps.live.com/?v=2&cid=74B8FFD299EDD840!106&encType=1
```

Moreover, you are able to get a GeoRSS representation of a Live Search Maps collection here:

```
http://maps.live.com/GeoCommunity.aspx?action=retrieverss&mkt=en-us~CCC
&cid={collection-id}
```

As you will see later in the chapter, GeoRSS is an XML vocabulary for embedding geographic information into RSS and Atom feeds. For a preview of what GeoRSS is, you can look at the following URL, which is shown in Listing 13-3.[18]

18. The file is cached at http://examples.mashupguide.net/ch13/ve.bookstores.georss.xml.

```
http://maps.live.com/GeoCommunity.aspx?action=retrieverss~CCC
&mkt=en-us&cid=74B8FFD299EDD840!10
```

Listing 13-3. GeoRSS of a Map from Microsoft's Live Search Maps

```
<rss version="2.0" xmlns:georss="http://www.georss.org/georss"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:Cml2GeoRssHelper="urn:Cml2GeoRssHelper">
  <channel>
```

```
    <title>Bookstores around Berkeley</title>
    <description>An example for mashupguide.net</description>
    <link>http://local.live.com/?v=2&amp;cid=74B8FFD299EDD840!106</link>
    <pubDate>Wed, 29 Aug 2007 23:32:51 GMT</pubDate>
    <item>
      <title>Moe's Books</title>
      <link>http://local.live.com/?v=2&amp;cid=74B8FFD299EDD840!106</link>
      <guid>74B8FFD299EDD840!128</guid>
      <pubDate>Wed, 29 Aug 2007 23:32:18 GMT</pubDate>
      <description>
      &lt;div style="padding:4px;"&gt; 2476 Telegraph Ave, Berkeley, CA
      &lt;/div&gt; &lt;br/&gt;
  </description>
      <georss:point>37.865523 -122.258492</georss:point>
    </item>
    <item>
      <title>Barnes &amp; Noble Booksellers</title>
      <link>http://local.live.com/?v=2&amp;cid=74B8FFD299EDD840!106</link>
      <guid>74B8FFD299EDD840!112</guid>
      <pubDate>Sat, 06 Jan 2007 16:47:10 GMT</pubDate>
      <description>
      &lt;div style="padding:4px;"&gt; 6060 El Cerrito Plz, El Cerrito, CA
      &lt;/div&gt; &lt;br/&gt;
  </description>
      <georss:point>37.899299 -122.300926</georss:point>
    </item>
    <item>
      <title>Cody's Books</title>
      <link>http://local.live.com/?v=2&amp;cid=74B8FFD299EDD840!106</link>
      <guid>74B8FFD299EDD840!107</guid>
      <pubDate>Wed, 25 Oct 2006 01:27:14 GMT</pubDate>
      <description>
      &lt;div style="padding:4px;"&gt; 1730 4th St, Berkeley, CA 94710
      &lt;/div&gt; &lt;br/&gt;
  </description>
      <georss:point>37.870975 -122.301021</georss:point>
    </item>
  </channel>
</rss>
```

Notice that each of the three bookstores is now associated with a `georss:point` element containing the latitude and longitude of the place (`georss` corresponds to the `http://www.georss.org/georss` namespace). For example, the element for Moe's Books is as follows:

```
<georss:point>37.865523 -122.258492</georss:point>
```

Notice a crucial difference between the GeoRSS originating from Live Search Maps and the XML of Yahoo! Local: the former contains the latitude and longitude of the bookstores, whereas the latter contains the addresses. We'll return to studying the implications of these differences later in the chapter.

Summary of Making Maps Without Programming

What can you conclude from these exercises? Mapbuilder.net is not so easy to use if you want to create a lot of markers but you don't already have those addresses laid out as CSV-formatted data. It would be nice to use a service like Yahoo! Local to pull up addresses and then pass the data into Mapbuilder.net. Live.local.com is surprisingly easy to use for building maps, but it's not easy to extract address information to create maps on competing map services. Google My Maps allows you to search for businesses and make maps of those results if you are content to make a Google map based on Google business data. Remember, however, that you can get a representation of the maps you make with Google My Maps as KML, which you can reuse elsewhere.

Data Exchange Formats

Before you study the details of specific APIs, you should understand the formats used to get data in and out of maps. Sometimes you can make a map by formatting what you want to place on a map in the right format. You already saw this approach in the previous section where I showed you how to transform the Yahoo! Local collection XML into CSV to feed to Mapbuilder.net. In the following sections, I'll examine some common data formats for online maps:

* CSV
* Microformats and other metatags
* GeoRSS
* KML

CSV

CSV stands for *comma-separated values*. It is a simple and widely supported format across many operating systems and applications. For tabular data, it is conceptually simpler and more compact than XML. In the case of Mapbuilder.net, you can upload the data for a set of markers using CSV format. For each marker, you can specify the caption, street address, city, state, and ZIP code. Mapbuilder.net can then geocode the addresses (that is, calculate the latitude and longitude of the address) to place them on a map. The built-in geocoding functionality of Mapbuilder.net is a major convenience for users.

For example, you can use a simple use of CSV by Geocoder.us (a service to convert U.S. addresses to latitude and longitude). The following:

```
http://rpc.geocoder.us/service/csv?address=2855+Telegraph+Ave.,+Berkeley,+CA
```

returns this:

```
37.858276,-122.260070,2855 Telegraph Ave,Berkeley,CA,94705
```

Microformats and Metatags for HTML

Some web sites have taken to embedding geographic information in HTML. As I will discuss in Chapter 18, microformats are little parcels of structured data that are seamlessly embedded in web pages—making them easily parsed by computer programs so that the data can be reused in other contexts. There are two relevant microformats in this context: geo and adr.

Consider the case of geotagged photos in Flickr, specifically the example I have already used in this book:

```
http://www.flickr.com/photos/raymondyee/18389540/
```

in which you will see the use of both the geo and adr microformats as instantiated in the following pieces of HTML, respectively:

```
<span class='geo' style='display:none'><span class='latitude'>37.8721</span>
<span class='longitude'>-122.257704</span></span>
```

```
<li id="li_location" class="Stats adr">
[....]
<span class='locality'>Oakland</span>, <span class='region'>California</span>
[....]
</li>
```

Caution Although the latitude and longitude are correct, the placement of the address in Oakland, California, is inaccurate unless Berkeley is being subsumed as part of Oakland. This inaccuracy doesn't take away from the syntactic correctness of the adr example.

You might notice also the use of two metatags that embed the latitude and longitude corresponding to the photo. The ICBM <meta> tag, which in this case is as follows, is documented at http://geourl.org/add.html:

```
<meta name="ICBM" content="37.8721, -122.257704">
```

You can learn more about the geo.position metatag, such as the following:

```
<meta name="geo.position" content="37.8721; -122.257704">
```

at http://geotags.com/geo/. Both the <meta> tags (located in the <head> section) are used to associate a latitude and longitude with a web page as a whole.

GeoRSS

GeoRSS (http://georss.org/) is a way of embedding location information within RSS 2.0, RSS 1.0, Atom 1.0, and potentially other XML formats. GeoRSS seems to be a standard—or at least an emerging one. In GeoRSS, you can represent points, lines, boxes, and polygons. Let's look in more detail at how to work with points.

GeoRSS is conceptually simple, but you might get confused by the fact that there are at least four ways to encode GeoRSS points in XML. The first two are the recommended encodings going forward, while the second set of two are considered legacy formats that are nonetheless widely used and therefore recommended for support. I list them here with examples:

* The *GeoRSS GML* encoding (`http://georss.org/gml`) wraps the `gml:pos` element in a `gml:Point` element within `georss:where`—where the `gml` prefix corresponds to `http://www.opengis.net/gml` and the `georss` prefix corresponds to `http://www.georss.org/georss`:

```
<georss:where>
  <gml:Point>
    <gml:pos>37.8721 -122.257704</gml:pos>
  </gml:Point>
</georss:where>
```

* The *GeoRSS Simple* encoding (`http://georss.org/simple`) uses a single `georss:point` element to contain the latitude and longitude, where `georss` corresponds to the same namespace as for the GeoRSS GML encoding (that is, `http://www.georss.org/georss`):

```
<georss:point>37.8721 -122.257704</georss:point>
```

* The *W3C Basic Geo* encoding (`http://georss.org/w3c`) uses `geo:Point` to wrap the `geo:lat` and `geo:long` elements—where the `geo` namespace prefix refers to `http://www.w3.org/2003/01/geo/wgs84_pos#._gml`:

```
<geo:Point>
<geo:lat>37.8721 </geo:lat>
<geo:long>-122.257704</geo:long>
</geo:Point>
```

* A common variant of the previous, which I call here the *Compact W3C Basic Geo* encoding, drops the enclosing `geo:Point`:

```
<geo:lat>37.8721</geo:lat>
<geo:long>-122.257704</geo:long>
```

If you refer to Listing 13-3, you will note that the Live Search Maps collection uses the *GeoRSS Simple* encoding. Let's also look at Flickr GeoFeed, which you can currently get for a given user here:

```
http://api.flickr.com/services/feeds/geo/?id={user-nsid} &lang=en-
us&format=rss_200
```

For example, the GeoFeed for Rev. Dan Catt (the driving force behind Flickr geotagging) is as follows:

```
http://api.flickr.com/services/feeds/geo/?id=35468159852@N01&lang=en-us~CCC
&format=rss_200
```

You can also get a GeoFeed for a given Flickr group here:

```
http://api.flickr.com/services/feeds/geo/?g={group-nsid} &lang=en-
us&format=rss_200
```

For example, the GeoFeed for the FlickrCentral group is as follows:

```
http://api.flickr.com/services/feeds/geo/?g=34427469792@N01&lang=en-us~CCC
&format=rss_200
```

Studying the GeoFeeds, you'll notice the use of two GeoRSS encodings, the GeoRSS Simple and W3C Basic Geo encodings, within the GeoFeeds:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
  xmlns:georss="http://www.georss.org/georss">
  <channel>
     [....]
    <item>
      <title>maybe not</title>
      <link>http://www.flickr.com/photos/solidether/1270523097/</link>
      [....]
      <georss:point>51.269057 12.345714</georss:point>
      <geo:Point>
        <geo:lat>51.269057</geo:lat>
        <geo:long>12.345714</geo:long>
      </geo:Point>
      [....]
    </item>
    [....]
  </channel>
</rss>
```

Yahoo!'s Use of GeoRSS and Yahoo! YMaps Extensions

When you read Yahoo!'s documentation at the following locations about its support for GeoRSS, you might get confused by the conflation of its own set of extensions to RSS 2.0 that use YMaps (corresponding to `http://api.maps.yahoo.com/Maps/V1/AnnotatedMaps.xsd` or `http://api.maps.yahoo.com/Maps/V2/AnnotatedMaps.xsd`—depending on which API you are using):

* `http://developer.yahoo.com/maps/simple/V1/reference.html` (for the Yahoo! Maps Simple API)
* `http://developer.yahoo.com/maps/georss/index.html` (for its Ajax API)

There are a whole bunch of tags, but I'll focus here on the ones for marking up addresses: `<ymaps:Address>`, `<ymaps:CityState>`, `<ymaps:Zip>`, and `<ymaps:Country>` to denote an address associated with an `<item>`.

The Yahoo! Simple API[19] lets you pass in the URL of an RSS 2.0 feed that has Compact W3C Basic Geo encoding and Yahoo!-specific extensions. See, for instance, the example given in the Yahoo! documentation:

```
http://api.maps.yahoo.com/Maps/V1/annotatedMaps?appid=YahooDemo~CCC
&xmlsrc=http://developer.yahoo.com/maps/sample.xml
```

This ability to use a mix of Geo and YMaps extensions reflects the ability of Yahoo! Maps to do the geocoding for you: you can still place a point on a map without knowing the latitude and longitude as long as you have an address that you can mark up with the appropriate `ymaps` tag. (We will look later at how to call on services to do explicit geocoding.) However, you must not conflate the YMaps extension with GeoRSS—they

are not the same. For instance, I don’t know of any applications outside of Yahoo! Maps that supports the YMaps extensions.
19. http://developer.yahoo.com/maps/simple/V1/reference.html

Let’s look at a working example of how to use the YMaps extension. Recall that you can use the Yahoo! Local API to get the XML for a collection here:

```
http://collections.local.yahooapis.com/LocalSearchService/V1/getCollection?~CC
C
appid={app-id}&collection_id={collection-id}
```

For example, the following is the URL for the collection of bookstores that I have assembled:

```
http://collections.local.yahooapis.com/LocalSearchService/V1/getCollection?~CC
C
appid={appp-id}&collection_id=1000014156
```

I have cached the results of the API call here:

```
http://examples.mashupguide.net/ch13/bookstores.yahoo.local.xml
```

You can convert this XML to RSS 2.0 with the YMaps extension either by hand or by using some XSLT code that I wrote for that purpose. What you get, which is cached here:

```
http://examples.mashupguide.net/ch13/yahoo.local.to.georss.xsl
```

is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:yahoo="urn:yahoo:travel"
     xmlns:ymaps="http://api.maps.yahoo.com/Maps/V1/AnnotatedMaps.xsd"
     xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
     version="2.0">
   <channel>
      <title>bookstores around Berkeley</title>
      <link>http://local.yahoo.com/collections?cid=1000014156</link>
      <description>some of my favorite bookstores around Berkeley.</description>
      <item>
         <title>Moe's Books</title>
         <link>http://local.yahoo.com/details?id=21518795&amp;stx=~CCC
&amp;csz=Berkeley+CA&amp;ed=xoDOxa16OSyYoswS6OvDhQk64pj4Q8RHG5PQhcSqprzxVT6mDHMe
zwfQ~CCC
2U244pugG4LDSdibA78iSw--</link>
         <description>
           * The Bay Area's Largest Selection of Used Scholarly Books
         </description>
         <ymaps:Address>2476 Telegraph Ave</ymaps:Address>
         <ymaps:CityState>Berkeley, CA</ymaps:CityState>
         <ymaps:Zip>94704</ymaps:Zip>
         <ymaps:Country>US</ymaps:Country>
      </item>
      <item>
         <title>Cody's Books</title>
         <link>http://local.yahoo.com/details?id=21512172&amp;stx=~CCC
```

```
&amp;csz=Berkeley+CA&amp;ed=3uqWba160SzFEqntYzu46yunejqBEmJnCBEi_I7QbD68sZTEVRYl
4WkO~CCC
GEf6alVIaEB3</link>
         <description/>
         <ymaps:Address>1730 4th St</ymaps:Address>
         <ymaps:CityState>Berkeley, CA</ymaps:CityState>
         <ymaps:Zip>94710</ymaps:Zip>
         <ymaps:Country>US</ymaps:Country>
      </item>
      <item>
         <title>Barnes &amp; Noble Booksellers</title>
         <link>http://local.yahoo.com/details?id=21414999&amp;stx=~CCC
&amp;csz=El+Cerrito+CA&amp;ed=Fo51gq160Sy24fPx_u7IvyZen3kxQq5wR9ZOi_Aos2J.pPlJ75
D_th~CCC
3K2MHtNCWF_V5k_nOq62ssy3I-</link>
         <description/>
         <ymaps:Address>6060 El Cerrito Plz</ymaps:Address>
         <ymaps:CityState>El Cerrito, CA</ymaps:CityState>
         <ymaps:Zip>94530</ymaps:Zip>
         <ymaps:Country>US</ymaps:Country>
      </item>
   </channel>
</rss>
```

Tip You can use the W3C XSLT services to perform online XSLT transformations at `http://www.w3.org/2001/05/xslt` and `http://www.w3.org/2005/08/online_xslt/` (for XSLT 2.0).

Go to the following URL to see the rendition of the location data in version 1 of Yahoo! Maps:

```
http://api.maps.yahoo.com/Maps/V1/annotatedMaps?appid={app-id}~CCC
&xmlsrc=http://examples.mashupguide.net/ch13/bookstores.georsss.xml
```

You can also pass the geocoded RSS 2.0 to the Ajax Yahoo! Maps.[20] One way to see this functionality at work is to follow these steps:

1. Adapt code from one of the Yahoo! examples[21] by centering it on the UC Berkeley campus and using your own Yahoo! API key.[22]
2. Bring up that example. Invoke the JavaScript Shell on it,[23] and type the following to load the RSS file (with the YMaps extensions) into the map:

```
map.addOverlay(~CCC
  new YGeoRSS('http://examples.mashupguide.net/ch13/bookstores.georsss.xml'));
```

20. http://developer.yahoo.com/maps/georss/index.html

21. http://developer.yahoo.com/maps/ajax/V3/ajaxexample1.html

22. http://examples.mashupguide.net/ch13/yahoo.map.berkeley.html

23. http://www.squarefree.com/shell/, a technique introduced in Chapter 8.

Caution If you plan to feed GeoRSS to the Yahoo! Maps APIs, use the Compact W3C Basic Geo encoding since that's the encoding implied by the documentation for Yahoo! Maps. Moreover, you may find the error messages from the Yahoo! Maps Simple API to be rather terse. I have found using Feedvalidator.org to be useful in debugging my GeoRSS.

GeoRSS in Virtual Earth

Virtual Earth also has the capacity to handle GeoRSS files using any of the four encodings, according to this:

`http://dev.live.com/virtualearth/sdk/Ref/HTML/WorkingWithLayers.htm`

Now we should be able to feed that file to Virtual Earth. Making a slight modification to some sample code[24] to read a cached version of my own Flickr GeoFeed,[25] I come up with the following demo code:

24. http://msdn2.microsoft.com/en-us/library/bb429606.aspx

25. http://examples.mashupguide.net/ch13/flickr.geofeed.xml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=5">
    </script>
    <script>
       var map = null;
       var layerid=1;

       function GetMap()
       {
          map = new VEMap('myMap');
          map.LoadMap();
       }

       function AddMyLayer(type)
       {
          var l = new VEShapeLayer();
          var txtSource = document.getElementById('txtSource');
          var veLayerSpec =
            new VEShapeSourceSpecification(type, txtSource.value, l);
          map.ImportShapeLayerData(veLayerSpec, onFeedLoad);
       }
       function onFeedLoad(feed)
       {
          feed0 = feed;
          alert('RSS or Collection loaded. There are '+feed.GetShapeCount()+
                ' items in this list.');
       }
    </script>
  </head>
  <body onload="GetMap();">
    <div id='myMap' style="position:relative; width:400px; height:400px;"></div>
    <input id="txtSource" type="text" value="flickr.geofeed.xml"
name="txtSource">
    <input id="loadFeed" type="button" value="Load RSS"
           onclick="AddMyLayer(VEDataType.GeoRSS);">
  </body>
</html>
```

This is available here:

`http://examples.mashupguide.net/ch13/virtualearth.flickrgeofeed.v5.html`

KML

Keyhole Markup Language (KML) "is an XML grammar and file format for modeling and storing geographic features such as points, lines, images, and polygons for display in Google Earth, Google Maps, and Google Maps for mobile."[26] Google's backing of KML makes it an important format for the exchange of geographic information.

KML has moved beyond its use in Google Earth alone. For instance, you can display KML files and export search results and one of your My Maps from Google Maps in KML. Other applications are beginning to support KML. For instance, you can get KML coming out of Yahoo! Pipes;[27] also, there is support for KML in Feed Validator.[28] KML is being shepherded through a standards process.[29] Google is advising people to use KML so that its geosearch can index KML—in KML 2.2, there is an `<attribution>` element. Google apparently will also index GeoRSS.

I'll cover the syntax KML in greater detail in the "Google Earth and KML" section. It's helpful, nonetheless, to see a simple example of a KML document so that you have something concrete to bounce off of:

26. http://earth.google.com/kml/

27. http://blog.pipes.yahoo.com/2007/05/02/pipes-adds-interactive-yahoo-maps-kml-support-and-more/

28. http://googleearthuser.blogspot.com/2007/05/feed-validator.html

29. http://geotips.blogspot.com/2007/04/kml-ogc.html

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Placemark id="berkeley">
    <description>Berkeley, CA</description>
    <name>Berkeley</name>
    <Point>
      <coordinates>-122.257704,37.8721,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Interoperability Among Formats: GeoRSS vs. KML

In commenting on the Where 2.0 conference, Benjamin Christen wrote this: "There are two important XML schemas covered today at Where 2.0—GeoRSS and KML.[30] Can we get these two formats—and others of importance—to work together?"[31] Mikal Maron provides an intriguing portrait of the relationship among various formats:[32]

> *There are of course other geodata formats in use, which deserve a look as alternatives to GeoRSS. KML is used in Google Earth, and loads of data layers have been published by an active community. However, KML is very tied to its application, with features specifically aimed for 3D spinny globes, and the spec is controlled by a single organization. GPX, for data interchange between GPS units, is again very tied to specifics of GPS units. GML is a feature-rich vocabulary for encoding geographic information, but its complexity has been daunting for unversed developers and its proper use misunderstood. GML is similar to RDF, defining a number of primitive objects that can be assembled into profiles for particular purposes. In fact, a GML profile for GeoRSS is a result of the new standard.*

Not surprisingly, Maron was involved in such interoperability efforts as MGeoRSS, an extension that "integrates basic GeoRSS support directly into Google Maps."[33] Someone from Google might have been listening to him—in March 2007, Google added native support for GeoRSS to its maps.

We're at a point now that since both GeoRSS and KML are getting a good amount of traction, some good interconversion utilities between them would be timely.

Creating Maps by API Programming

In the following sections, I will discuss the APIs of various popular online services, specifically, Google, Yahoo!, and Microsoft. In the previous section, I discussed data formats with an eye to making maps without any (or much) programming. In the following sections, we will do a bit of programming.

I'll now summarize what these maps can do in general. The four covered here all allow you to do the following:

* Embed an Ajax-based map, to which you can add custom locations with pop-up windows
* Geocode addresses (translate an address to latitude and longitude), at least for U.S. and Canadian addresses
* Show the maps at various zoom levels and of various types (road, aerial, or hybrid)
* Add lines to the maps to represent features such as driving directions

30. http://www.oreillynet.com/conferences/blog/2006/06/georss_and_kml.html
31. http://www.ogleearth.com/2006/05/georss_is_here.html is a good summary of some of the issues.
32. http://xtech06.usefulinc.com/schedule/paper/56
33. http://brainoff.com/gmaps/mgeorss.html

With a bit of copying and pasting, you can get working examples of each of the maps. You can then modify them incrementally. Using the DOM Inspector and the

JavaScript Shell, you can even make changes to live working examples within the browser. I will also use those mechanisms to highlight important capabilities and functions of the maps.

Google Maps API

Let's look at the Google Maps API. We will start with how to embed a Google map using the Google Maps API. The online documentation on how to get started with the maps at the Google web site is good.[34]

We'll set up a simple map and then use the JavaScript Shell to work with a live map so that you can invoke a command and see an immediate response. The intended effect is that you see the widgets as dynamic programs that respond to commands, whether that command comes in a program or from you entering the commands one by one.

Note This tutorial on the Google Maps API is essentially the same as that in Chapter 8 and is repeated here for your convenience.

Getting Started with Google Maps and the JavaScript Shell

We will use the Google Maps API to make a simple map:

1. Make sure you have a public web directory to host your map and know the URL of that directory. Any Google map that uses the free, public API needs to be publicly visible.

2. Go to the sign-up page for a key to access Google Maps.[35] You will need a key for any given domain in which you host Google Maps. (It is through these keys that Google regulates the use of the Google Maps API.)

3. Read the terms of service,[36] and if you agree to them, enter the URL directory on the host that you want to place your test file. For example, in my case, the URL is `http://examples.mashupguide.net/ch13/`. Note that key.

4. Copy and paste the HTML code into your own page on your web-hosting directory. You should get something like my own example:[37]

34. http://www.google.com/apis/maps/documentation/#Introduction

35. http://www.google.com/apis/maps/signup.html

36. http://www.google.com/apis/maps/terms.html

37. http://www.google.com/maps/api_signup?url=http%3A%2F%2Fexamples.mashupguide.net%2Fch13%2F

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=<API_KEY>"
      type="text/javascript"></script>
    <script type="text/javascript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

5. Now make one modification to the example by removing the `var` keyword in front of `map` to make it a global variable that is thus accessible to the JavaScript Shell. That is, change this:

```
var map = new GMap2(document.getElementById("map"));
```

to the following:

```
map = new GMap2(document.getElementById("map"));
```

to expose the `map` object to the JavaScript Shell utility.[38]

6. Invoke the JavaScript Shell for your map by hitting the JavaScript Shell bookmarklet in the context of your map. Type the code fragments in the following steps, and see what happens. (Note that another approach is to modify your code directly with these code fragments and reload your page.) These actions use version 2 of the Google Maps API.[39]

7. To return the current zoom level of the map (which goes from 0 to 17, with 17 being the most detailed), type the following command (the response from the JavaScript Shell is shown right after the code):

```
map.getZoom()
```

```
13
```

38. http://examples.mashupguide.net/ch13/google.map.1.html

39. http://www.google.com/apis/maps/documentation/reference.html#GMap2

8. To obtain the latitude and longitude of the center of the map, do this:

```
map.getCenter()
```

```
(37.4419, -122.1419)
```

9. To center the map around the Campanile for UC Berkeley, use this:

```
map.setCenter(new GLatLng(37.872035,-122.257844), 13);
```

10. You can pan to that location instead:

```
map.panTo(new GLatLng(37.872035,-122.257844));
```

11. To add a small map control (to control the zoom level), run these two commands:

```
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
```

12. To turn GMap keyboard navigation on, use this:

```
window.kh = new GKeyboardHandler(map);
```

```
[object Object]
```

13. To fully zoom out the map, use this:

```
map.setZoom(0)
```

14. To zoom in all the way (maximum zoom level may go from 15 to 17), do this:

```
map.setZoom(17)
```

15. To set the variable `maptypes` to an array holding three objects, use this:

```
maptypes = map.getMapTypes()
```

```
[object Object],[object Object],[object Object]
```

16. To get the name of the first entry in `maptypes`, use this:

```
map.getMapTypes()[0].getName()40
```

```
Map
```

17. To get the current map type, you can get the object and the name of that type object:

```
map.getCurrentMapType()
```

```
[object Object]
```

40. 1 corresponds to satellite, while 2 corresponds to the hybrid map type.

```
map.getCurrentMapType().getName()
```

```
Map
```

18. To set maptype to satellite, do this:

```
map.setMapType(maptypes[1]);
```

19. You can zoom one level in and out if you are not already at the max or min zoom level:

```
map.zoomIn();
map.zoomOut();
```

20. To make an overlay, try this:

```
point = new GLatLng (37.87309185260284, -122.25508689880371);
```

```
(37.87309185260284, -122.25508689880371)
```

```
marker = new GMarker(point);
```

```
[object Object]
```

```
map.addOverlay(marker);
```

21. To make something happen when you click the marker, do this:

```
GEvent.addListener(marker, ‘click’, function() {
marker.openInfoWindowHtml(‘hello’); });
```

```
[object Object]
```

22. To add a new layer of pins from a GeoRSS feed, use this:

```
map.addOverlay(new GGeoXml(‘http://api.flickr.com/services/feeds/geo/~CCC
?id=48600101146@N01&lang=en-us&format=rss_200’));
```

There are many more things to explore, such as polylines and overlays and draggable points. To learn more, I certainly recommend the “Google Maps API: Introduction” document.[41] Note that “this documentation is designed for people familiar with JavaScript programming and object-oriented programming concepts. You should also be familiar with Google Maps from a user’s point of view.”

41. http://www.google.com/apis/maps/documentation/#Introduction

Yahoo! Maps API

The Yahoo! Maps can be programmed with its API. You can find the core documentation, “Yahoo! Maps Web Services: Introducing the Yahoo! Maps APIs,” on the Yahoo! web site.[42] As I described in the previous sections, the Simple API[43] is useful to get started with because the declarative approach does not involve any JavaScript programming but creates only the right XML file.

Note If you want to learn about the Flash APIs for Yahoo! Maps, which are outside the scope of this book, please refer to the official documentation at `http://developer.yahoo.net/maps/flash/index.html`.

Getting Started with Yahoo! Maps and the JavaScript Shell

In this exercise, I will present a step-by-step introduction to the Ajax APIs for Yahoo! Maps:[44]

1. Apply for a Yahoo! application key.[45] You are told not to use the `appid` for the example code in the Yahoo! documentation (which is `YahooDemo`).

2. Copy and paste the following code for your web site:[46]

```
<html>
  <head>
    <script type=”text/javascript”
src=”http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=<API_Key>”></script>
    <style type=”text/css”>
      #mapContainer {
      height: 500px;
      width: 80%;
      }
    </style>
  </head>
<body>
  <div id=”mapContainer”></div>
  <script type=”text/javascript”>
    // Create a lat/lon object
    var myPoint = new YGeoPoint(37.4041960114344,-122.008194923401);
    // Create a map object
    var map = new YMap(document.getElementById(‘mapContainer’));
    // Display the map centered on a latitude and longitude
    map.drawZoomAndCenter(myPoint, 3);

    // Add map type control
    map.addTypeControl();

    // Set map type to either of: YAHOO_MAP_SAT YAHOO_MAP_HYB YAHOO_MAP_REG
    map.setMapType(YAHOO_MAP_SAT);

    //Get valid map types,
    //returns array [YAHOO_MAP_REG, YAHOO_MAP_SAT, YAHOO_MAP_HYB]
    var myMapTypes = map.getMapTypes();
  </script>
</body>
</html>
```

42. http://developer.yahoo.net/maps/

43. http://developer.yahoo.net/maps/simple/index.html

44. http://developer.yahoo.net/maps/ajax/index.html

45. https://developer.yahoo.com/wsregapp/index.php

46. I have my code at http://examples.mashupguide.net/ch13/yahoo.map.simple.1.html.

3. Invoke the JavaScript Shell for your map by hitting the JavaScript Shell bookmarklet in the context of your map. Type the code fragments in the following steps, and see what happens. (If you don't use your own map, try the JavaScript Shell on the example on the Yahoo! web site.)[47]

4. To get the zoom level, use this:

```
map.getZoomLevel()
```

```
3
```

5. To get the center location, specifically, the latitude and longitude, use this:

```
map.getCenterLatLon()
```

```
[object Object]
```

```
props(map.getCenterLatLon())
```

```
Fields: Lat, Lon
Methods of prototype: distance, equal, getRad, greater, middle, pointDiff,
setgeobox, valid
```

```
map.getCenterLatLon().Lat
```

```
37.4041960114344
```

```
map.getCenterLatLon().Lon
```

```
-122.008194923401
```

47. http://developer.yahoo.com/maps/ajax/V3/ajaxexample1.html

6. To set the zoom level (15 for largest scale, 1 for most zoomed in), use this:

```
map.setZoomLevel(15)
map.setZoomLevel(1)
```

7. To move the map to a new center (in this case, the UC Berkeley campus), do this:

```
p = new YGeoPoint(37.87309185260284, -122.25508689880371)
```

```
[object Object]
```

```
map.panToLatLon(p)
```

8. To add some navigation and zoom controls, use this:

```
map.addPanControl();
map.addZoomLong();
```

9. To add a marker, labeled *H*, use the following:

```
marker = new YMarker(p);
```

```
[object Object]
```

```
marker.addLabel("H");
map.addOverlay(marker);
```

10. To make a click event invoke a pop-up, use this:

```
function onSmartWinEvent() {var words = "Yeah Yahoo maps!"; ~CCC
marker.openSmartWindow(words); }
YEvent.Capture(marker, EventsList.MouseClick, onSmartWinEvent);
```

11. To add a marker to a specific address, in this case, 2195 Hearst Ave, you can use either of the following three alternatives:

```
map.addMarker("2855 Telegraph Ave, Berkeley, CA");
```

or

```
map.addOverlay(new YMarker("2855 Telegraph Ave, Berkeley, CA "));
```

or

```
marker = new YMarker("2855 Telegraph Ave, Berkeley, CA ");
marker.addLabel("Apress");
map.addOverlay(marker);
 marker.reLabel("<b>hello</b>");
```

12. To add a GeoRSS feed (that uses either the Yahoo! YMaps extensions or the Compact W3C Basic Geo encoding) to the map, use this:

```
map.addOverlay(new YGeoRSS('http://examples.mashupguide.net/ch13/~CCC
bookstores.georsss.xml'));
```

These examples do not exhaust the Yahoo! Ajax API, which has features such as polylines and more complete overlay functionality.

Microsoft's Live Search Maps/Virtual Earth

Microsoft's Live Search Maps, the company's latest offering in online maps, has gone by quite a few other names (Windows Live Map and Windows Live Local). The name of the service is also not to be confused with Virtual Earth, the name Microsoft has given to the technology that "powers" Live Search Maps. None of this technology is to be confused with MSN Maps, also run by Microsoft.[48]

The Virtual Earth Map control is an Ajax widget, well documented at the following locations:

* The official central place for the Microsoft Virtual Earth docs.[49]
* The Virtual Earth Interactive SDK[50] is a great place to learn about Virtual Earth because it combines a live demo with relevant source code and links to the reference documentation. (Think of it as a counterpart to the Flickr API Explorer.)

Getting Started with Virtual Earth and the JavaScript Shell

1. Copy and paste the following code (this is the simplest piece of code given at the Virtual Earth Interactive SDK):[51]

```
<html>
   <head>
      <title>ve.map.1.html</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <script
src="http://dev.virtualearth.net/mapcontrol/v5/mapcontrol.js">
      </script>
      <script>
      var map = null;

      function GetMap()
      {
         map = new VEMap('myMap');
         map.LoadMap();
      }
      </script>
   </head>
   <body onload="GetMap();">
      <div id="myMap"
           style="position:relative; width:400px; height:400px;"></div>
   </body>
</html>
```

48. http://mappoint.msn.com/

49. http://dev.live.com/virtualearth

50. http://dev.live.com/virtualearth/sdk/

51. http://examples.mashupguide.net/ch13/ve.map.1.html

2. Invoke the JavaScript Shell for your map by hitting the JavaScript Shell bookmarklet in the context of your map. Type the code fragments in the following steps, and see what happens.

3. To get the zoom level, use this:

```
map.GetZoomLevel()
```

```
4
```

4. To zoom in and out, use this:

```
map.ZoomIn();
map.ZoomOut();
```

5. To get the style of the map, use this:

```
map.GetMapStyle()
```

```
r
```

6. To set the map style (a = aerial, r = road, h = hybrid, and o = bird's-eye), use this:

```
map.SetMapStyle('a');
```

3D Aspects of Virtual Earth

One cool distinguishing feature of the Virtual Earth is its 3D mode, accessible via JavaScript. This 3D mode is akin to wrapping Google Earth functionality into Google Maps. The 3D mode is available if you are running Firefox or Internet Explorer version 6 or 7 on Windows and have the appropriate 3D add-ons installed. The requirements are documented here:

```
http://msdn2.microsoft.com/en-us/library/bb429547.aspx
```

Continuing the exercise, you can put the map into 3D mode with the following command in the JavaScript Shell:

```
map.SetMapMode(VEMapMode.Mode3D);
```

Remember to use the Virtual Earth Interactive SDK (`http://dev.live.com/virtualearth/sdk/`) to learn about the other capabilities in Virtual Earth, including working with shapes and driving directions.

Geocoding

A common task in using online maps is to geocode addresses—that is, converting street addresses to the corresponding latitude and longitude. In the following sections, I will walk through the basics of geocoding in Yahoo!, Google, Geocoder.us, and Virtual Earth maps, which is enough to get you started. For the following examples, I use the address of Apress, which is 2855 Telegraph Ave., Berkeley, CA.

Caution There are subtleties that I won't go in detail about: the precision and accuracy of the APIs, dealing with ambiguities in the addresses, and which geocoder is best for a given geographic location.

Yahoo! Maps

Yahoo! provides a REST geocoding method here:

`http://developer.yahoo.com/maps/rest/V1/geocode.html`

whose base URL is `http://api.local.yahoo.com/MapsService/V1/geocode` and whose parameters include the `appid` to identify your application and two ways of identifying the address:

* A combination of `street`, `city`, `state`, and `zip`:

```
http://api.local.yahoo.com/MapsService/V1/geocode?appid={app-id}~CCC
&street=2855+Telegraph+Ave.&city=Berkeley&state=CA
```

* `location`, which is free text that consists of one string that holds a combination of `street`, `city`, `state`, and `zip`. The `location` string has priority for determining the placement of this:

```
http://api.local.yahoo.com/MapsService/V1/geocode?appid={app-id}~CCC
&location=2855+Telegraph+Ave.%2C+Berkeley%2C+CA
```

When you use these two methods to geocode the location of the Apress office, you get the same result. Using `street`, `city`, and `state` returns this:

```
<?xml version="1.0"?>
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns="urn:yahoo:maps"
           xsi:schemaLocation="urn:yahoo:maps
http://api.local.yahoo.com/MapsService/V1/GeocodeResponse.xsd">
  <Result precision="address">
    <Latitude>37.858377</Latitude>
    <Longitude>-122.259171</Longitude>
    <Address>2855 TELEGRAPH AVE</Address>
    <City>BERKELEY</City>
    <State>CA</State>
    <Zip>94705-1128</Zip>
    <Country>US</Country>
  </Result>
</ResultSet>
```

Note the following characteristics of the output:

* You get the same results with the two methods in this case.
* You can compare the output address and the input address to make sure the geocoder is interpreting the address the way you think it should be.
* The default output format is XML when no `output` parameter is specified.
* You get the latitude and longitude in the `<Latitude>` and `<Longitude>` elements, respectively.

A good way to see how the API behaves is to try various parameters. See what happens in the following cases:

* Specify only `city=Berkeley` to get several results corresponding to the cities that go by the name of Berkeley.[52]
* Use the `output=php` option to get serialized PHP.[53]
* Enter a nonexistent street address for a given city.[54]

Geocoder.us

Geocoder.us provides a free gecoding service for U.S. addresses. Refer to Chapter 7 for a detailed discussion of the REST and SOAP interfaces to the Geocoder.us API, which is documented here:

```
http://geocoder.us/help/
```

Let's calculate the latitude and longitude for the Apress office with different aspects of the Geocoder.us service:

* The Geocoder.us user interface, invoked with this:

```
http://geocoder.us/demo.cgi?address=2855+Telegraph+Ave.%2C+Berkeley%2C+CA
```

shows that the latitude and longitude of the address is the following:

```
(37.858276, -122.260070)
```

* The CSV interface, invoked with this:

```
http://rpc.geocoder.us/service/csv?address=2855+Telegraph+Ave.%2C+Berkeley%2C+CA
```

returns the following:

```
37.858276,-122.260070,2855 Telegraph Ave,Berkeley,CA,94705
```

* The REST interface, through this:

```
http://geocoder.us/service/rest/?address=2855+Telegraph+Ave.%2C+Berkeley%2C+CA
```

returns the following:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <geo:Point rdf:nodeID="aid33483656">
    <dc:description>2855 Telegraph Ave, Berkeley CA 94705</dc:description>
    <geo:long>-122.260070</geo:long>
    <geo:lat>37.858276</geo:lat>
  </geo:Point>
</rdf:RDF>
```

Notice the use of the W3 Basic Geo encoding for the latitude and longitude in the response.

52. http://api.local.yahoo.com/MapsService/V1/geocode?appid={app-id}&city=Berkeley

53. http://api.local.yahoo.com/MapsService/V1/geocode?appid={app-id}&location=350+5th+Ave,+New+York,+NY&output=php

54. http://api.local.yahoo.com/MapsService/V1/geocode?appid={app-id}&location=350000+main+Street,+Berkeley,+CA

Google Geocoder

The Google Geocoder provides two interfaces: a REST interface and a JavaScript-accessible interface:

`http://www.google.com/apis/maps/documentation/#Geocoding_Examples`

I'll cover each in turn.

REST Interface

We'll first look at the REST method, whose base URL is `http://maps.google.com/maps/geo` and whose parameters are as follows:

* `q` is the address to geocode.
* `key` is your API key.[55]
* `output` is the format of the output—one of `xml`, `kml`, `csv`, or `json`.

Let's look at some example output. The `xml` and `kml` output for 2855 Telegraph Ave., Berkeley, CA, produces the same body (listed next) but different `Content-Type` headers ("text/xml" and "application/vnd.google-earth.kml+xml"), respectively:[56]

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
  <Response>
    <name>2855 Telegraph Ave., Berkeley, CA.</name>
    <Status>
      <code>200</code>
      <request>geocode</request>
    </Status>
    <Placemark id="p1">
      <address>2855 Telegraph Ave, Berkeley, CA 94705, USA</address>
      <AddressDetails Accuracy="8"
xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
        <Country>
          <CountryNameCode>US</CountryNameCode>
          <AdministrativeArea>
            <AdministrativeAreaName>CA</AdministrativeAreaName>
            <SubAdministrativeArea>
              <SubAdministrativeAreaName>Alameda</SubAdministrativeAreaName>
              <Locality>
                <LocalityName>Berkeley</LocalityName>
                <Thoroughfare>
                  <ThoroughfareName>2855 Telegraph Ave</ThoroughfareName>
                </Thoroughfare>
                <PostalCode>
                  <PostalCodeNumber>94705</PostalCodeNumber>
                </PostalCode>
              </Locality>
            </SubAdministrativeArea>
          </AdministrativeArea>
        </Country>
      </AddressDetails>
      <Point>
        <coordinates>-122.259310,37.858517,0</coordinates>
      </Point>
    </Placemark>
  </Response>
</kml>
```

55. You can get your key at http://www.google.com/apis/maps/signup.html. ABQIAAAAdjiS7YH6Pzk2Nrli02b5xxR10RG5t-vK3TwPKbpNUO2c5sYb4RTmySs_TEFzYvlZrCaYJKlmTzJ5lA is the key for http://examples.mashupguide.net/ch13/.

56. http://maps.google.com/maps/geo?q=2855+Telegraph+Ave.%2C+Berkeley%2C+CA.&output=xml&key=

ABQIAAAAdjiS7YH6Pzk2Nrli02b5xxR10RG5t-vK3TwPKbpNUO2c5sYb4RTmySs_TEFzYvlZrCaYJKlmTzJ5lA

Caution Even though the XML output does include a `</Placemark>` element, which is valid KML, the `</Response>` element in the output precludes the output from being valid KML.

Here is the `json` output for convenient use in JavaScript:[57]

```
{"name":"2855 Telegraph Ave., Berkeley, CA.","Status":{"code":200,"request":
"geocode"},"Placemark":[{"id":"p1","address":"2855 Telegraph Ave, Berkeley,
CA 94705, USA","AddressDetails":{"Country":{"CountryNameCode":"US",
"AdministrativeArea":{"AdministrativeAreaName":"CA","SubAdministrativeArea":
{"SubAdministrativeAreaName":"Alameda","Locality":{"LocalityName":"Berkeley",
"Thoroughfare":{"ThoroughfareName":"2855 Telegraph Ave"},"PostalCode":
{"PostalCodeNumber":"94705"}}}}},"Accuracy": 8},"Point":{"coordinates":
[-122.259310,37.858517,0]}}]}
```

JavaScript Interface

As for the JavaScript methods available in the Google mapping system, consult the documentation provided by Google,[58] which points to using the `GClientGeocoder` object. You can use the JavaScript Shell to see this object in action:

1. Open the simple Google map from the previous section.[59]
2. In the JavaScript Shell, set up the address and an instance of `GClientGeocoder`:

```
address = "2855 Telegraph Ave., Berkeley, CA"
```

```
2855 Telegraph Ave., Berkeley, CA
```

```
geocoder = new GClientGeocoder();
```

```
[object Object]
```

3. Compose a call to `GClientGeocoder.getLatLng`, which takes an address and a callback function, which in turn takes the point geocoded from the address:

```
geocoder.getLatLng(address, function(point) {
    if (!point) {
      alert(address + " not found");}
    else {
      map.setCenter(point, 13);
      var marker = new GMarker(point);
      map.addOverlay(marker);
      marker.openInfoWindowHtml(address);
    }
  }
);
```

If you get rid of the whitespace in the function, you can easily invoke it in the JavaScript Shell:

```
geocoder.getLatLng( address, function(point) {
  if (!point) { alert(address + " not found");}
  else {
    map.setCenter(point, 13);
    var marker = new GMarker(point);
    map.addOverlay(marker);
    marker.openInfoWindowHtml(address);
  }
} );
```

You will then see that Google Maps adds an overlay marking 2855 Telegraph Ave., Berkeley, CA (as shown in Figure 13-1).

57. http://maps.google.com/maps/geo?q=2855+Telegraph+Ave.%2C+Berkeley%2C+CA.&output=json&key=

ABQIAAAAdjiS7YH6Pzk2Nrli02b5xxR10RG5t-vK3TwPKbpNUO2c5sYb4RTmySs_TEFzYvlZrCaYJKlmTzJ5lA

58. http://www.google.com/apis/maps/documentation/#Geocoding_JavaScript

59. http://examples.mashupguide.net/ch13/google.map.1.html

Insert 858Xf1301.tif

Figure 13-1. Invoking the JavaScript Google Geocoder with the JavaScript Shell

Virtual Earth

Virtual Earth provides geocoding functionality in the `VEMap.Find` method of version 5 of the Virtual Earth SDK,[60] which takes two parameters: a location string and a callback function. Let's illustrate this at work with the JavaScript Shell:

1. Open the basic Virtual Earth example.[61]

2. Let's create a callback function that pops up an alert with the latitude/longitude of the found locations:

```
function onFoundResults (ShapeLayer,FindResult,Place,HasMore) {
  html = "";
  for (x=0; x<Place.length; x++) {
    html = html + Place[x].LatLong + "";
  }
  alert (html);
}
```

60. http://msdn2.microsoft.com/en-US/library/bb429645.aspx

61. http://examples.mashupguide.net/ch13/ve.map.1.html

3. In the JavaScript Shell, type the following:

```
address = "2855 Telegraph Ave., Berkeley, CA"
```

```
2855 Telegraph Ave., Berkeley, CA
```

```
function onFoundResults (ShapeLayer,FindResult,Place,HasMore) {
  html = "";
  for (x=0; x<Place.length; x++) {html = html + Place[x].LatLong + "";}
  alert (html);
}
map.Find(null,address,null,null,null,null,null,null,null,null, onFoundResults);
```

Note The last line with the many `null` parameters might look surprising, but it is an officially recommended invocation from http://msdn2.microsoft.com/en-us/library/bb545008.aspx.

4. You will see an alert that pops up the latitude and longitude of the address.

I packaged this logic in a simple example:[62]

```
<html>
  <head>
    <title>VE Map showing VEMap.Find (ve.map.find.html)</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script
src="http://dev.virtualearth.net/mapcontrol/v5/mapcontrol.js"></script>
    <script>
    var map = null;

    function onFoundResults (ShapeLayer,FindResult,Place,HasMore) {
      html = "";
      //alert("Place: " + Place);
      for (x=0; x<Place.length; x++) {
        html = html + Place[x].LatLong + "";
      }
      alert (html);
    }

    function GetMap()
    {
       map = new VEMap('myMap');
       map.LoadMap();

       address = "2855 Telegraph Ave., Berkeley, CA";
       map.Find(null,address,null,null,null,null,null,null,null,
                null, onFoundResults);
    }
    </script>
  </head>
  <body onload="GetMap();">
    <div id='myMap' style="position:relative; width:400px; height:400px;"></div>
  </body>
</html>
```

62. http://examples.mashupguide.net/ch13/ve.map.find.html

Geocoding Non-U.S. Addresses

As I have implied in my previous examples, the Google, Yahoo!, Microsoft, and Geocoder.us geocoders are built to handle American addresses. What if you want to geocode street-level addresses in other countries? It would be great to have a single geocoding API that would return a reliable latitude and longitude for an address from anywhere in the world. That service doesn't currently exist. Here, however, are some starting points to geocoding addresses from around the world:

* The Google geocoder can geocode addresses in countries listed here:

`http://code.google.com/support/bin/answer.py?answer=62668&topic=12266`

The list currently includes Austria, Australia, Belgium, Brazil, Canada, The Czech Republic, Denmark, Finland, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Luxembourg, The Netherlands, New Zealand, Poland, Portugal, Singapore, Spain, Sweden, Switzerland, Taiwan, the United Kingdom, and the United States. Note the caveat that the accuracy of the results can vary per country.

* The best list I could find for Yahoo!'s coverage is here:

`http://ylocalblog.com/blog/2007/05/16/yahoo-maps-global-rollout-gets-a-new-look-`
~CCC
`%e2%80%93-and-a-new-platform/`

This blog entry lists the following countries in Western Europe as having "complete coverage": Austria, Belgium, Denmark, Finland, Germany, Great Britain, Luxembourg, Netherlands, Ireland, Norway, Portugal, Spain, Sweden, Switzerland, France, and Italy.

* Consult the following for lists of other geocoders:

`http://groups.google.com/group/Google-Maps-API/web/resources-non-google-`
~CCC`geocoders`

```
http://mapki.com/wiki/FAQs#Geocoding
```

Google Earth and KML

Google Earth (`http://earth.google.com/`) is a virtual globe, which means it is a desktop environment that simulates the three-dimensional aspects of the earth. It runs on Windows, Mac OS X, and Linux. Google Earth is a cool application, rightfully described as *immersive*. There are other virtual globes,[63] but I wouldn't be surprised if Google Earth remains the dominant virtual globe platform for geodata sharing.[64]

Google Earth is also a great mashup platform. What makes it so?

* The three-dimensional space of a planet is an organizing framework that is easy to understand—everyone knows his or her place in the world, so to speak.
* KML—the XML data format for getting data in and out of Google Earth is easy to read and write.
* There are other APIs to Google Earth, including a COM interface in Windows and an AppleScript interface in Mac OS X.

Displaying and Handling KML As End Users

I introduced KML earlier in the chapter but reserved a full discussion of it in the context of Google Earth. The main reason for this organizational choice is that although KML is steadily growing beyond its origins as the markup language for Keyhole, the precursor to Google Earth, the natural home for KML remains Google Earth. Google Earth is the fullest user interface for displaying and interacting with KML. You can also use it to create KML. At the same time, since Google Earth is not the only tool for working with KML, I'll describe some useful tips for using those other tools.

A good and fun way to start with KML is to download and install Google Earth and to use it to look at a variety of KML files. Here are some sources of KML:

* The Google Earth Gallery (`http://earth.google.com/gallery/index.html`)
* The Google Earth Community (`http://bbs.keyhole.com/ubb/ubbthreads.php/Cat/0`)

It turns out that Google Maps and Flickr are also great sources of KML. After walking you through a specific example to demonstrate the mechanics of interacting with KML, I will describe how to get KML out of Flickr and Google Maps—and how to use Google Maps to display KML.

Let me walk you through the mechanics with one example:

```
http://maps.google.com/maps/ms?f=q&hl=en&geocode=&ie=UTF8&msa=0~CCC
&msid=116029721704976049577.0000011345e68993fc0e7&z=14&om=1
http://maps.google.com/maps/ms?f=q&msa=0&output=kml&msid=116029721704976049577.
~CCC
0000011345e68993fc0e7
```

63. http://en.wikipedia.org/wiki/Virtual_globe

64. http://www.technologyreview.com/read_article.aspx?ch=specialsections&sc=personal&id=17537 makes the argument that Google Earth will be exactly that dominant platform.

Downloading KML into Google Earth

1. Make sure you have Google Earth installed. You can download it from here:

```
http://earth.google.com/
```

2. After you have installed Google Earth, learn how to navigate the interface. At the least, you should get comfortable with typing addresses or business names, causing the Google interface to go to those places. Also learn how to use the Save to My Places functionality to create collections of individual items and how to change the properties of individual items, including the latitude, longitude, icon, and view of the item. Finally, you should be to be able to get KML corresponding to the collection.[65] With the KML in hand and an understanding of what a collection looks like in Google Earth, you are in a good position not only to read KML but also to write KML.

3. The classic way that most Google Earth users interact with KML is through clicking a link that causes a KML file to be downloaded and fed into Google Earth. (I'll cover the technical mechanism for how that happens later in the chapter.) Here, I'll walk through one such example of a KML file that you can download.

   Go to the map of bookstores I created with Google My Maps:

```
http://maps.google.com/maps/ms?f=q&msa=0&msid=116029721704976049577.~CCC

0000011345e68993fc0e7&z=14&om=1
```

4. Click the KML link, which is as follows:

```
http://maps.google.com/maps/ms?f=q&om=1&ie=UTF8&msa=0&output=nl~CCC

&msid=116029721704976049577.0000011345e68993fc0e7
```

5. If your browser and Google Earth are set up in the typical configuration (in which Google Earth is registered to handle files with a `Content-Type` header of "application/vnd.google-earth.kml+xml"), you will be prompted to let Google Earth open the downloaded KML. If you accept, the collection of points representing the bookstores is loaded into Temporary Places. Double-clicking the link of the collection spins the markers into view in Google Earth. See Figure 13-2 to see the Google Maps collection displayed in Google Earth.

65. To start with, you should know the default locations of your stored KML files: C:\Documents and Settings\[USERNAME]\Application Data\Google\GoogleEarth\myplaces.kml (Win32) and ~/Library/Google Earth/myplaces.kml (OS X).

Insert 858Xf1302.tif

Figure 13-2. A Google Maps collection displayed in Google Earth

Now that you have downloaded a KML file into Google Earth, let's look at other tools that are useful for your study of KML.

Google Maps As a KML Renderer

You can use Google Maps to display the contents of a KML file. The easiest way to do so is to go to the Google Maps page (`http://maps.google.com`) and enter the URL of the KML file as though it were an address or other search term. Such a query results in the following URL:

```
http://maps.google.com?q={kml-url}
```

For example, feeding the KML for the bookstore map back into Google Maps, we get the following:

```
http://maps.google.com/maps?q=http:%2F%2Fmaps.google.com%2Fmaps%2Fms%3Ff%3Dq%26o
m%3D~CCC
1%26ie%3DUTF8%26msa%3D0%26output%3Dnl%26msid%3D116029721704976049577.0000011345e
6899~CCC
3fc0e7
```

I have found using Google Maps to be an incredibly useful KML renderer. First, you can test KML files without having access to Google Earth. Second, you can let others look at the content of KML files without requiring them to have Google Earth installed. You should be aware, however, of two caveats in using Google Earth to render KML:

* Google Maps does not implement KML in total—so don't expect to replace Google Earth with Google Maps for working with KML.
* Google Maps caches KML files that you render with it. That is, if you are using Google Maps to test the KML that you are changing, be aware that Google Maps might not be reading the latest version of your KML file.

KML from Flickr

You can now use Flickr to learn more about KML. Currently, although there's no official documentation, you can refer to Rev. Dan Catt's weblog entry to learn some of the details about getting KML out of Flickr:

```
http://geobloggers.com/archives/2007/05/31/flickr-kml-and-a-stroll-down-memory-
lane/
```

The following KML feeds contain at most 20 entries. You have two choices about the format to use:

* `format=kml_nl` for the KML network link that refreshes periodically to show the latest photos. (I will discuss the KML `<NetworkLink>` element in a moment.)
* `format=kml` for the static KML that contains the data about the locations.

So, anywhere I write `format=kml_nl`, you can substitute `format=kml`.

When you are looking at these feeds, it's helpful to remember that in addition to using Google Earth as a KML viewer, you can use Google Maps, which I find very convenient. Just drop the URL for the KML file into the search box for Google Maps, and you can have the KML file displayed in Google Maps. For instance, you can take the KML for the 20 most recent geotagged photos in Flickr, like so:

```
http://api.flickr.com/services/feeds/geo/?format=kml_nl
```

and drop it into Google Maps, which you can access here:

```
http://maps.google.com/maps?q=http:%2F%2Fapi.flickr.com%2Fservices%2Ffeeds%2Fgeo
%2F%~CCC
3Fformat%3Dkml_nl
```

Currently, you can get a KML feed for an individual user with this:

```
http://api.flickr.com/services/feeds/geo/?id={user-nsid}&format=kml_nl
```

For example, here's the feed for Rev. Dan Catt:

```
http://api.flickr.com/services/feeds/geo/?id=35468159852@N01&format=kml_nl
```

You can get the KML feed for a group here:

```
http://api.flickr.com/services/feeds/geo/?g={group-nsid}&format=kml_nl
```

For example, here's the KML feed for the Flickr Geotagging group:

```
http://api.flickr.com/services/feeds/geo/?g=94823070@N00&format=kml_nl
```

You can get KML feeds for a tag, such as `flower`:

```
http://api.flickr.com/services/feeds/geo/&tags=flower&format=kml_nl
```

You can get KML feeds for locations. Here are some examples:

```
http://api.flickr.com/services/feeds/geo/us/?format=kml_nl
http://api.flickr.com/services/feeds/geo/us/ca/berkeley/?format=kml_nl
http://api.flickr.com/services/feeds/geo/uk/london/?format=kml_nl
http://api.flickr.com/services/feeds/geo/ca/on/toronto/?format=kml_nl
http://api.flickr.com/services/feeds/geo/FR/%C3%8Ele-de-
France/Paris?format=kml_nl
http://api.flickr.com/services/feeds/geo/cn/beijing/?format=kml_nl
http://api.flickr.com/services/feeds/geo/cn/%E5%8C%97%E4%BA%AC/?format=kml_nl
```

Note that the last two links are both for Beijing ().

You can do a combined search on a user, location, and tags. For example, to get Raymond Yee's photos in Berkeley, California, tagged with `flower`, use this:

```
http://api.flickr.com/services/feeds/geo/us/ca/berkeley/?id=48600101146@N01~CC
C
&tags=flower&format=kml_nl
```

Caution Look for official documentation on KML in Flickr to see how it evolves beyond its early beginnings.

KML

Although KML at its heart is a simple dialect of XML, Google is steadily adding features to do more and more through KML. The home for KML documentation is here:

```
http://code.google.com/apis/kml/documentation/
```

Don't overlook the file of KML samples that the documentation refers to, because the examples are very useful:

`http://code.google.com/apis/kml/documentation/KML_Samples.kml`

You can have these rendered in Google Maps:

`http://maps.google.com/maps?q=http%3A%2F%2Fkmlscribe.googlepages.com%2F`*~CCC*
`SamplesInMaps.kml`

The goal in this section is to get you started with how to read and write KML. Let's start with a simple example of KML that contains a single `<Placemark>` element, whose associated `<Point>` element is located at the Campanile of the UC Berkeley campus:[66]

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Placemark id="berkeley">
    <description>Berkeley, CA</description>
    <name>Berkeley</name>
    <Point>
      <coordinates>-122.257704,37.8721,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

The `<Point>` element defines the position of the placemark's name and icon. You can read the file into Google Earth to display it or use Google Maps to render it:

`http://maps.google.com/maps?q=http:%2F%2Fexamples.mashupguide.net%2Fch13%2F`*~CC*
C
`berkeley.simple.kml`

You can validate the KML using the techniques described here:

`http://googlemapsapi.blogspot.com/2007/06/validate-your-kml-online-or-`
`offline.html`

For instance, you can feed to the file to Feedvalidator.org, which validates KML (in addition to RSS and Atom feeds):

`http://feedvalidator.org/check.cgi?url=http%3A%2F%2Fexamples.mashupguide.net%2F`
~CCC
`ch13%2Fberkeley.simple.kml`

Using your favorite XML validator, you can also validate the KML against the XML Schema for KML 2.1 located here:

`http://code.google.com/apis/kml/schema/kml21.xsd`

66. http://examples.mashupguide.net/ch13/berkeley.simple.kml

Adding a View to a Placemark: LookAt and Camera

The previous KML document defined a specific point for positioning the placemark's name and icon but did not specify a viewpoint for the placemark. Given that you can zoom around the virtual globe from many angles, you won't be surprised that KML allows you to define a viewpoint associated with a placemark. There are two ways to do so: the `<LookAt>` element and the `<Camera>` element (defined in KML 2.2 and later). You will find excellent introductory documentation for these two elements here:

```
http://code.google.com/apis/kml/documentation/cameras.html
```

Here I use a series of examples to illustrate the central difference between `<LookAt>` and `<Camera>`: `<LookAt>` specifies the viewpoint in terms of the location being viewed, whereas `<Camera>` specifies the viewpoint in terms of the viewer's location and orientation. I have gathered the examples in a single KML file:

```
http://examples.mashupguide.net/ch13/berkeley.campanile.evans.kml
```

which you can view through Google Earth or Google Maps here:

```
http://maps.google.com/maps?f=q&hl=en&geocode=&q=http:%2F%2Fexamples.mashupguide
.net~CCC
%2Fch13%2Fberkeley.campanile.evans.kml&ie=UTF8&ll=37.872507,-122.257565~CCC
&spn=0.006581,0.01133&z=17&om=1
```

or here:

```
http://tinyurl.com/38oawy
```

I recommend loading the KML into Google Earth and in Google Maps so that you can follow along. See Figure 13-3 for how this KML file looks in Google Maps. Notice that there are three `<Placemark>` elements for two buildings: the Campanile, which sits almost due south of Evans Hall, and Evans Hall, both on the UC Berkeley campus.

Insert 858Xf1303.tif

Figure 13-3. KML for the Campanile and Evans Hall in Google Maps

The first placemark is placed at the Campanile and uses a `<LookAt>` element to specify a viewpoint for the Campanile (see Figure 13-4 for a rendering of this placemark in Google Earth):

```
<Placemark id="Campanile">
  <name>Campanile</name>
  <description><![CDATA[Campanile viewed from the South, using  <LookAt>
(range=200,
tilt=45, heading=0)]]></description>
  <LookAt>
    <longitude>-122.257704</longitude>
    <latitude>37.8721</latitude>
    <altitude>0</altitude>
    <altitudeMode>relativeToGround</altitudeMode>
    <range>200</range>
    <tilt>45</tilt>
    <heading>0</heading>
  </LookAt>
  <Point>
    <coordinates>-122.257704,37.8721,0</coordinates>
  </Point>
</Placemark>
```

Insert 858Xf1304.tif

Figure 13-4. View of the Campanile in Google Earth defined by the `<LookAt>` element

In addition to the `<Point>` element, which specifies a marker to click for the placemark, the KML defines a point of focus whose latitude and longitude are `37.8721` and `-122.257704` and whose altitude is 0 meters (relative to the ground). The distance between this point of focus and the viewer is specified in meters by the `range` element. (In this example, the virtual camera of the viewpoint is 200 meters away from the point.) Two further parameters control the orientation the viewpoint:

* `tilt` specifies the angle of the virtual camera relative to an axis running perpendicular to the ground. In other words, 0 degrees means you are looking straight down at the ground; 90 degrees means you're looking at the horizon. In the case of our example, `tilt` is 45 degrees, which you can pick out from Figure 13-4. (`tilt` is limited to the range of 0 to 90 degrees.) Note that the default value is 0 degrees, which is, in fact, the only value that is supported in Google Maps.
* `heading` (which ranges from 0 to 360 degrees) specifies the geographic direction you are looking at. Zero degrees, the default value, means the virtual camera is pointed north. In this example, `heading` is indeed 0—we are looking at the Campanile from the south. We can see Evans Hall to the north. In Google Maps, the rendered heading is fixed to 0 degrees—north is always up.

Now let's consider a second placemark, which appears as Figure 13-5 when rendered in Google Earth:

```
<Placemark id="Evans">
  <name>On the roof of  Evans Hall  Looking at Campanile</name>
  <description><![CDATA[Looking south from Evans to Campanile, using <Camera>
(heading=180,  tilt=90, roll=0)]]></description>
  <Camera>
    <longitude>-122.2578687854004</longitude>
    <latitude>37.87363451913904</latitude>
    <altitude>50</altitude>
    <altitudeMode>relativeToGround</altitudeMode>
    <heading>180</heading>
    <tilt>90</tilt>
    <roll>0</roll>
  </Camera>
  <Point>
    <coordinates>-122.2578687854004,37.87363451913904,0</coordinates>
  </Point>
</Placemark>
```

Insert 858Xf1305.tif

Figure 13-5. View of the Campanile in Google Earth defined by the `<Camera>` element

You set up the `<Camera>` element by specifying the location and orientation of the viewer instead of the point of focus. The latitude/longitude for the camera corresponds to Evans Hall, a point north of the Campanile. The altitude (50 m) is roughly roof height for Evans Hall. Let's look at the three angles that are part of a `<Camera>` element definition:

* `heading` is set to 180 degrees, meaning the virtual camera is pointing south. Thus, the Campanile is in view.
* `tilt` is set to 90 degrees, meaning it is looking parallel to the ground. (In contrast to the tilt for `<LookAt>`, which is constrained to a value between 0 and 90 degrees, the tilt for `<Camera>` can go from 0 to 180 degrees.) It can even be negative, which results in an upside-down view. This difference in the range for `tilt` allows a `<Camera>` element to be aimed at the sky, whereas a `<LookAt>` element could at most be aimed at the horizon but not above it.
* `roll` is set to 0 degrees, which is the default value. You can look at the third placemark defined here:

`http://examples.mashupguide.net/ch13/berkeley.campanile.evans.kml`

in which roll is 45 degrees to get a sense of the effect that roll has on a view (refer to Figure 13-6 to see the third placemark).

Insert 858Xf1306.tif

Figure 13-6. View of the Campanile in Google Earth defined by the `<Camera>` element with a 45-degree roll

The final KML file uses a folder to group the three `<Placemark>` elements, and it associates a `<LookAt>` element for the folder to give the folder a view.

Programming Google Earth via COM and AppleScript

In addition to controlling Google Earth through the user interface or through feeding it KML, you can also program Google Earth. In Windows, you can do so through the COM interface and on Mac OS X via AppleScript.[67] Little information is publicly available for these APIs. In the following sections, I'll show you some small samples to get you started and to show you some of the capabilities of the API.

COM Interface

You can find documentation of the COM interface to Google Earth here:

`http://earth.google.com/comapi/index.html`

The following is a small sample Python snippet (running on Windows with the Win32 extension) to load the previous KML example, highlight each of the placemarks, and render their respective views in turn. Note that the code uses the `OpenKmlFile` method to read a local file so that it can then get features using the `GetFeatureByHref` method.

```
# demonstrate the Google Earth COM interface
#
import win32com.client
ge =  win32com.client.Dispatch("GoogleEarth.ApplicationGE")
```

```
fn =
r'D:/Document/PersonalInfoRemixBook/examples/ch13/berkeley.campanile.evans.kml'
ge.OpenKmlFile(fn,True)

features = ['Campanile', 'Evans', 'Evans_Roll']

for feature in features:s
    p = ge.GetFeatureByHref(fn + "#" + feature)
    p.Highlight()
    ge.SetFeatureView(p,0.1)
    raw_input('hit to continue')
```

The following is a Python code example to demonstrate the use of `SetCameraParams` to set the current view of Google Earth. Notice how the parameters correspond to those of the `<LookAt>` element. At some point, perhaps, the Google Earth COM interface will support the parameters corresponding to the `<Camera>` element in KML.

```
# demonstrate the Google Earth COM interface
#
import win32com.client
ge =  win32com.client.Dispatch("GoogleEarth.ApplicationGE")

# send to UC Berkeley Campanile
lat = 37.8721
long = -122.257704

#altitude in meters
altitude = 0

#
http://earth.google.com/comapi/earth_8idl.html#5513db866b1fce4e039f09957f57f8b7
# AltitudeModeGE { RelativeToGroundAltitudeGE = 1, AbsoluteAltitudeGE = 2 }
altitudeMode = 1

#range in meters; tilt in degrees, heading in degres
range = 200
tilt = 45
heading = 0 # aka azimuth

#set how fast to send Google Earth to the view
speed = 0.1

ge.SetCameraParams(lat,long,altitude,altitudeMode,range,tilt,heading,speed)
```

67. http://www.ogleearth.com/2006/09/google_earth_fo_6.html

AppleScript Interface to Google Earth

In Mac OS X, you would use AppleScript to control Google Earth or something like `appscript`, an Apple event bridge that allows you to write Python scripts in place of AppleScript:

`http://appscript.sourceforge.net/`

Here's a little code segment in AppleScript to get you started—it will send you to the Empire State Building:

```
tell application "Google Earth"
  activate
  set viewInfo to (GetViewInfo)
  set dest to {latitude:57.68, longitude:-95.4, distance:1.0E+5,
tilt:90.0, azimuth:180}
  SetViewInfo dest speed 0.1
end tell
```

This moves the view to 57.68 N and 95.4 degrees W.

Using appscript, the following Python script also steers Google Earth in Mac OS X:

```
#!/Library/Frameworks/Python.framework/Versions/Current/bin/pythonw
from appscript import *
ge = app("Google Earth")
#h = ge.GetViewInfo()
h = {k.latitude: 36.510468818615237, k.distance: 5815328.0829986408,~CCC
k.azimuth: ~CCC
10.049582258046936, k.longitude: -78.864908202209094, k.tilt:~CCC
3.0293063358608456e-14}
ge.SetViewInfo(h,speed=0.5)
```

Mapstraction and OpenLayers

In this chapter, I've covered how to use some of the major mapping APIs: Google Maps, Yahoo!, MapQuest, and Microsoft. It would be convenient to be able to not worry about the differences among the maps and easily switch among the various maps. That's the promise of a mapping abstraction library such as Mapstraction (http://mapstraction.com). We'll have to wait and see how and whether it is widely used to gauge the library's effectiveness.

Along a different vein is OpenLayers (http://www.openlayers.org/), which is defined as follows:

> *A pure JavaScript library for displaying map data in most modern web browsers, with no server-side dependencies. OpenLayers implements a (still-developing) JavaScript API for building rich web-based geographic applications, similar to the Google Maps and MSN Virtual Earth APIs, with one important difference—OpenLayers is free software, developed for and by the open source software community.*

You can try OpenLayers in FlashEarth (http://www.flashearth.com/). Go to the site, and select OpenLayers. You might have to zoom out sufficiently to see any tiles (for example, go to http://www.flashearth.com/?lat=38.417308&lon=-122.271821&z=9.9&r=0&src=ol). You can also check out other examples in the OpenLayers gallery at http://www.openlayers.org/gallery/.

An Integrative Example: Showing Flickr Pictures in Google Earth

In this section of this chapter, I'll walk you through an example that mashes up Flickr, Google Earth, and Google Maps. Specifically, I'll show you how to query Flickr for public geotagged photos and convert the response to KML that can then be channeled to either Google Earth or Google Maps.

You can see the program that combines this functionality here:

```
http://examples.mashupguide.net/ch13/flickrgeo.php
```

And you can see the PHP code here:

```
http://examples.mashupguide.net/ch13/flickrgeo.php.txt
```

The `flickrgeo.php` script described in this chapter is the same code as that in Chapter 10.

What you will see is a basic HTML form aimed at a user who already understands the parameters for the `flickr.photos.search` method (`http://examples.mashupguide.net/ch13/flickrgeo.php`). There are two differences between the parameters used for Flickrgeo and the Flickr API:

* The `bbox` parameter used by the Flickr API is broken out into four individual parameters for Flickrgeo: `lat0`, `lon0` for the southwest corner and `lat1`, `lon1` for the northeast corner of the bounding box. This approach will be familiar to you from Chapter 10, which focuses in detail on how to query Flickr for geotagged photos.
* Flickrgeo has an `o_format` parameter to set the requested output. Recognized values are `html` (for the simple user interface), `rest` (to return the Flickr API in REST format), `json` (to return the results as JSON), `nl` (to return a KML NetworkLink), and `kml` (to return the photo data as KML). The appropriate value for `format` is passed to `flickr.photos.search`.

When the Flickrgeo form loads, `o_format` is set by default to `html` so that you can use the form to enter some values and see some search results rendered as HTML. For example, the following:

```
http://examples.mashupguide.net/ch13/flickrgeo.php?user_id=48600101146%40N01&tag
s=~CCC&text=&lat0=37.817785166068&lon0=-122.34375&lat1=37.926190569376&lon1=-
122.17208862305~CCC&page=1&per_page=10&min_upload_date=820483200&extras=geo&o
_format=html
```

or

```
http://tinyurl.com/2nbjbb
```

displays my public geotagged photos in the Berkeley area. You can change `o_format` to `json` and `rest` to get JavaScript and XML versions of this data, respectively; Flickrgeo just passes back the data that comes from Flickr.

If you set `o_format` to `kml`, you get the results as a KML feed. For example, here's a KML feed of my public geotagged photos in the Berkeley area:

```
http://examples.mashupguide.net/ch13/flickrgeo.php?user_id=48600101146%40N01&tag
s=~CCC&text=&lat0=37.817785166068&lon0=-122.34375&lat1=37.926190569376&lon1=-
```

```
122.17208862305&~CCCpage=1&per_page=10&min_upload_date=820483200&extras=geo&o
_format=kml
```

or

```
http://tinyurl.com/36hu2j
```

which you can see on Google Maps (see Figure 13-7):[68]

```
http://maps.google.com/maps?f=q&hl=en&geocode=&q=http:%2F%2Fexamples.mashupguide
.net~CCC%2Fch13%2Fflickrgeo.php%3Fuser_id%3D48600101146%2540N01%26tags%3D%26t
ext%3D%26lat0%~CCC3D37.81778516606 8%26lon0%3D-
122.34375%26lat1%3D37.926190569376%26lon1%3D-
122.17208862305~CCC%26page%3D1%26per_page%3D10%26min_upload_date%3D820483200%
26extras%3Dgeo%26o_format%~CCC3Dkml&ie=UTF8&z=14&om=1
```

or

68. http://tinyurl.com/2fmhh5

Insert 858Xf1307.tif

Figure 13-7. Flickr photos displayed in Google Maps via a KML feed

If you set `o_format` to `nl`, you get a KML NetworkLink that enables you to use Google Earth to interact with Flickrgeo. That is, if you change your viewpoint on Google Earth, Google Earth will send to Flickrgeo the parameters of your new viewpoint to get pictures for that new viewpoint.

Hence, Flickrgeo does four major tasks to pull off the mashup:

1. Querying Flickr for geotagged photos
2. Converting Flickr results from a single `flickr.photos.search` into the corresponding KML
3. Generating a KML NetworkLink to feed your actions in Google Earth to Flickrgeo
4. Knitting together the various possibilities for `o_format` into one script

For the first topic, I refer you to Chapter 10 for a detailed discussion and remind you that the essential point is that you use the `bbox` parameter to specify a bounding box, add `geo` to the extras parameter to get the latitude and longitude, and set a minimum upload time to something like 820483200 (for January 1, 1996 Pacific Time) to coax some photos from the API when you are not searching on tags or text.

For the final topic, you can see the logic of what is done by studying the source code. Next I'll discuss the topics of KML NetworkLink and generating KML from `flickr.photos.search` results.

KML NetworkLink

So far I have shown you how to write a KML document and render it with Google Earth and Google Maps. In many situations, including the mashup I create here, it's extremely helpful to get information flowing from Google Earth back to the program that generated the KML in the first place. Foremost among that data would be the current viewpoint of the user. If you as the generator of the KML know the region that the user is focused on, you can generate data that would be visible in that viewpoint.

That's the purpose of the `<NetworkLink>` element in KML. Here I'll show you how to use it by example. Load the following into Google Earth:

```
http://examples.mashupguide.net/ch13/hello.world.networklink.kml
```

You will see a pin appear in the middle of your current viewpoint announcing the current time. If you click the pin, you'll see a list of parameters corresponding to the current viewpoint. If you change the viewpoint and wait a couple of seconds, the pin is refreshed to be located in the center of the new viewpoint. How does this happen?

Let's look first at this:

```
http://examples.mashupguide.net/ch13/hello.world.networklink.kml
```

which is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
 <kml xmlns="http://earth.google.com/kml/2.2">
   <NetworkLink>
     <flyToView>0</flyToView>
     <name>Hello World</name>
     <open>1</open>
     <visibility>1</visibility>
     <Link>

<href>http://examples.mashupguide.net/ch13/hello.world.networklink.php</href>
       <viewRefreshMode>onStop</viewRefreshMode>
       <viewRefreshTime>2</viewRefreshTime>~CCC
<viewFormat>bboxWest=[bboxWest]&amp;bboxSouth=[bboxSouth]&amp;bboxEast=[bboxEast
]~CCC
&amp;bboxNorth=[bboxNorth]&amp;lookatLon=[lookatLon]&amp;lookatLat=[lookatLat]~
CCC
&amp;lookatRange=[lookatRange]&amp;lookatTilt=[lookatTilt]&amp;lookatHeading=~C
CC
[lookatHeading]&amp;lookatTerrainLon=[lookatTerrainLon]&amp;lookatTerrainLat=~C
CC
[lookatTerrainLat]&amp;lookatTerrainAlt=[lookatTerrainAlt]&amp;cameraLon=[camera
Lon]~CCC
&amp;cameraLat=[cameraLat]&amp;cameraAlt=[cameraAlt]&amp;horizFov=[horizFov]~C
CC
&amp;vertFov=[vertFov]&amp;horizPixels=[horizPixels]&amp;vertPixels=[vertPixels]
~CCC
&amp;terrainEnabled=[terrainEnabled]</viewFormat>
     </Link>
   </NetworkLink>
 </kml>
```

A KML NetworkLink defines how often refreshing occurs or the conditions under which it happens. I'll discuss what a refresh actually does in the next paragraph. There are two modes of refreshing that can be specified in a NetworkLink. The first, based on time, uses the `<refreshMode>` and `<refreshInterval>` elements. With those tags, you can, for instance, set a refresh to happen every ten seconds. For the KML I present here, I use refreshing based on changes in the viewpoint, which are parameterized by two tags: `<viewRefreshMode>` and `<viewRefreshTime>`. If you consult the KML documentation, you'll see that one choice for `<viewRefreshMode>` is `onStop`—which means a refresh event happens once the viewpoint stops changing for the amount of time specified by the `<viewRefreshTime>` element—in this case two seconds.

So, what happens during a refresh event? Google Earth does an HTTP `GET` request on the URL specified by the `href` element with parameters specified in the `<viewFormat>` element, which contains something akin to a URI template. That is, Google Earth substitutes the values that correspond to the viewpoint at the time of the refresh event for the value templates such as `[bboxWest]`, `[bboxSouth]`, `[bboxEast]`, and so on. Consult the documentation for a comprehensive list of parameters supported in KML. The template I specify here has the complete current list of parameters. Note that there is no requirement that the parameter names match the naming scheme given by Google. In fact, you should match the parameter names to the ones recognized by the script specified by the `href` element.

Let's now turn to the script here:

```
http://examples.mashupguide.net/ch13/hello.world.networklink.php
```

to see how the HTTP `GET` request is processed. Here's the PHP code:

```
<?php
// get the time
$timesnap = date("H:i:s");

// for clarity, place each coordinate into a clearly marked bottom-left
// or top-right variable

$bboxWest  = isset($_GET['bboxWest']) ? $_GET['bboxWest'] : "-180.0";
$bboxSouth  = isset($_GET['bboxSouth']) ? $_GET['bboxSouth'] : "-90.0";
$bboxEast  = isset($_GET['bboxEast']) ? $_GET['bboxEast'] : "180.0";
$bboxNorth  = isset($_GET['bboxNorth']) ? $_GET['bboxNorth'] : "90.0";

// calculate the approx center of the view -- note that this is inaccurate
// if the user is not looking straight down
$userlon = (($bboxEast - $bboxWest)/2) + $bboxWest;
$userlat = (($bboxNorth - $bboxSouth)/2) + $bboxSouth;

$response = '<?xml version="1.0" encoding="UTF-8"?>';
$response .= '<kml xmlns="http://earth.google.com/kml/2.2">';
$response .= '<Placemark>';
$response .= "<name>Hello at: $timesnap</name>";

# calculate all the parameters

$arg_text = "";
foreach ($_GET as $key => $val) {
  $arg_text .= "<b>{$key}</b>:{$val}<br>";
}

$description_text = $arg_text;
$description = "<![CDATA[{$description_text}]]>";
$response .= "<description>{$description}</description>";

$response .= '<Point>';
$response .= "<coordinates>$userlon,$userlat,0</coordinates>";
```

```
$response .= '</Point>';
$response .= '</Placemark>';
$response .= '</kml>';
# set $myKMLCode together as a string
 $downloadfile="myKml.kml"; # give a name to appear at the client
 header("Content-disposition: attachment; filename=$downloadfile");
 header("Content-Type: application/vnd.google-earth.kml+xml; charset=utf8");
 header("Content-Transfer-Encoding: binary");
 header("Content-Length: ".strlen($response));
 header("Pragma: no-cache");
 header("Expires: 0");
echo $response;
?>
```

hello.world.networklink.php reads all the parameters that are passed to it and displays them to the user. This is accomplished by generating the KML for a <Placemark> element with a <description> element with all the parameters.

Let's return to how I use the KML NetworkLink in flickrgeo.php. In that script, I needed to generate <NetworkLink> elements that looked like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <NetworkLink>
    <flyToView>0</flyToView>
    <name>Pictures from Flickr</name>
    <description>
      <![CDATA[<a href='http://examples.mashupguide.net/ch13/flickrgeo.php?
text=stop+sign&page=1&per_page=10&min_upload_date=820483200&extras=geo%2Clicense
%2Cowner_name%2Cicon_server%2Ctags&o_format=html'>Search Something
Different</a>]]>
    </description>
    <open>1</open>
    <visibility>1</visibility>
    <Link>
      <href>http://examples.mashupguide.net/ch13/flickrgeo.php?
text=stop+sign&amp;page=1&amp;per_page=10&amp;min_upload_date=820483200
&amp;extras=geo&amp;o_format=kml</href>
      <viewRefreshMode>onStop</viewRefreshMode>
      <viewRefreshTime>3</viewRefreshTime>
<viewFormat>lat0=[bboxSouth]&amp;lon0=[bboxWest]&amp;lat1=[bboxNorth]
&amp;lon1=[bboxEast]</viewFormat>
    </Link>
  </NetworkLink>
</kml>
```

This is a <NetworkLink> element for generating refreshable KML on photos that match a full-text search for stop sign. Notice how the viewpoint is passed from Google Earth to http://examples.mashupguide.net/ch13/flickrgeo.php by the viewFormat element:

```
lat0=[bboxSouth]&amp;lon0=[bboxWest]&amp;lat1=[bboxNorth]&amp;lon1=[bboxEast]
```

Generating the KML for the Photos

The following excerpt of KML shows the structure of the KML that `flickrgeo.php` generates to display the photos in Google Earth or Google Maps (I have put some KML elements in bold that I have yet to introduce.):

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Style id="118550863">
      <IconStyle>
        <Icon>
       <href>http://farm1.static.flickr.com/56/118550863_1b8f5a26aa_s.jpg</href>
        </Icon>
      </IconStyle>
    </Style>
    [....]
    <Folder>
      <name>Flickr Photos</name>
      <description>Total Number of Photos available: 72&amp;nbsp;&lt;~CCC
a href='http://examples.mashupguide.net/ch13/flickrgeo.php?~CCC
user_id=48600101146%40N01&amp;lat0=37.75976100792328&amp;lon0=-122.~CCC
4470955684774&amp;lat1=37.95649244418595&amp;lon1=-122.1471302328438~CCC
&amp;page=1&amp;per_page=10&amp;min_upload_date=820483200&amp;extras=geo%2Clicen
se~CCC
%2Cowner_name%2Cicon_server%2Ctags&amp;o_format=kml'&gt;KML&lt;/a&gt;&amp;nbsp;&
lt;~CCC
a href='http://maps.google.com?q=http%3A%2F%2Fexamples.mashupguide.net%2F~CCC
ch13%2Fflickrgeo.php%3Fuser_id%3D48600101146%2540N01%26lat0%3D37.75976100792328%
26~CCC
lon0%3D-122.4470955684774%26lat1%3D37.95649244418595%26lon1%3D-122.~CCC
1471302328438%26page%3D1%26per_page%3D10%26min_upload_date%3D820483200%26extras%
3D~CCC
geo%252Clicense%252Cowner_name%252Cicon_server%252Ctags%26o_format%3Dkml'&gt;GMa
p~CCC
&lt;/a&gt;</description>
      <Placemark>
        <name>shrink wrap car</name>
        <description>
          <![CDATA[<a
href='http://www.flickr.com/photos/48600101146@N01/118550863'>
<img src='http://farm1.static.flickr.com/56/118550863_1b8f5a26aa.jpg'></a>]]>
        </description>
        <LookAt>
          <longitude>-122.300915</longitude>
          <latitude>37.898562</latitude>
          <altitude>0</altitude>
          <altitudeMode>relativeToGround</altitudeMode>
          <range>2000</range>
          <tilt>0</tilt>
          <heading>0</heading>
        </LookAt>
        <styleUrl>#118550863</styleUrl>
        <Point>
```

```
          <coordinates>-122.300915,37.898562,0</coordinates>
        </Point>
      </Placemark>
      [....]
  </Folder>
  </Document>
</kml>
```

Note the following features of the KML:

* There is a `<Placemark>` element for each photo, whose `name` element holds the title for the photo. In the `<description>` element is HTML for the medium-sized version of the photo. The latitude and longitude, drawn from the geo information provided by Flickr, goes into two places: the coordinates element for the `<Point>` element and a `<LookAt>` view.
* Each `<Placemark>` element is tied to a `<Style>` element to generate custom icons for each photo. The icon is the square version of the Flickr photo. The association is made through the `<styleUrl>` element.
* There is a `<Folder>` element that groups all the `<Placemark>` elements. The `<description>` element for the `<Folder>` element contains links to the KML itself and to a Google Map showing this KML. These links provide you with a way of getting hold of what you are seeing in Google Earth.

The flickrgeo.php Code

Here's an edited listing of the `flickrgeo.php` code:

```
<?php
# flickrgeo.php
# copyright Raymond Yee, 2007
# http://examples.mashupguide.net/ch13/flickrgeo.php

# xmlentities substitutes characters in $string that can be expressed
# as the predefined XML entities.

function xmlentities ($string)
{ return str_replace (
           array ( '&', '"', "'", '<', '>' ),
           array ( '&amp;' , '&quot;', '&apos;' , '&lt;' , '&gt;' ),
           $string );
}

# converts an associative array representing form parameters
# and values into the request part of a URL.

function form_url_params($arg_list, $rid_empty_value=TRUE) {
    $list = array();

    foreach ($arg_list as $arg => $val) {
     if (!($rid_empty_value) || (strlen($val) > 0)) {
        $list[] = $arg . "=" . urlencode($val);
     }
    }
    return join("&",$list);
  }

# a simple wrapper around flickr.photos.search for public photos.
# It deals a request for either the Flickr REST or JSON formats

class flickrwrapper {
  protected $api_key;

  public function __construct($api_key) {
    $this->api_key = $api_key;
  }

  # generic method for retrieving content for a given URL.
  protected function getResource($url){
    $chandle = curl_init();
    curl_setopt($chandle, CURLOPT_URL, $url);
    curl_setopt($chandle, CURLOPT_RETURNTRANSFER, 1);
    $result = curl_exec($chandle);
    curl_close($chandle);

    return $result;
  }

  # returns an HTTP response body and headers
  public function search($arg_list) {
    # attach API key
    $arg_list['api_key'] = $this->api_key;

    # attach parameters specific to the format request, which is either JSON or
REST.
    $format = $arg_list["format"];
    if ($format == "rest") {
      $url = "http://api.flickr.com/services/rest/?method=flickr.photos.search&"
.
form_url_params($arg_list);
      $rsp = $this->getResource($url);
      $response["body"] = $rsp;
      $response["headers"] = array("Content-Type"=>"application/xml");
      return $response;
    } elseif ($format == "json") {
      $arg_list["nojsoncallback"] = 1;
      $url = "http://api.flickr.com/services/rest/?method=flickr.photos.search&"
.
form_url_params($arg_list);
      $rsp = $this->getResource($url);
      $response["headers"] = array("Content-Type"=>"text/javascript");
      $response["body"] = $rsp;
      return $response;
    }
  } // search
} //flickrwrapper
```

```
class flickr_html {

# generates a simple form based on the parameters
# and values of the input associative array $arg_array
# uses $path as the target of the form's action attribute

  public function generate_form($arg_array, $path) {
    $form_html = "";
    foreach ($arg_array as $arg => $default) {
      $form_html .= <<<EOT
{$arg}:<input type="text" size="20" name="{$arg}" value="{$default}"><br>
EOT;
    }

    $form_html = <<<EOT
<form action="{$path}" method="get">
{$form_html}<br>
<input type="submit" value="Go!">
</form>
EOT;

    return $form_html;
  } //generate_form

  # generates a simple HTML representation of the results of
flickr.photos.search
  public function html_from_pics($rsp) {

    $xml = simplexml_load_string($rsp);
    #print_r($xml);
    #var_dump($xml);
    $s = "";
    $s .= "Total number of photos: " . $xml->photos['total'] . "<br>";

    # http://www.flickr.com/services/api/misc.urls.html
    # http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
    foreach ($xml->photos->photo as $photo) {
      $farmid = $photo['farm'];
      $serverid = $photo['server'];
      $id = $photo['id'];
      $secret = $photo['secret'];
      $owner = $photo['owner'];
      $thumb_url =

"http://farm{$farmid}.static.flickr.com/{$serverid}/{$id}_{$secret}_t.jpg";
      $page_url = "http://www.flickr.com/photos/{$owner}/{$id}";
      $image_html= "<a href='{$page_url}'><img src='{$thumb_url}'></a>";
      $s .= $image_html;
    }
    return $s;
  }
} // flickr_html

# a class to handle conversion of Flickr results to KML
```

```
class flickr_kml {

# helper function to create a new text node with $string that is wrapped by an
# element named by $childName -- and then attach the whole thing to $parentNode.
# allow for a namespace to be specified for $childName

  protected function attachNewTextNode($parentNode,
                                       $childName,$childNS="",$string="") {
    $childNode = $parentNode->appendChild(new DOMElement($childName,$childNS));
    $childNode->appendChild(new DOMText($string));
    return $childNode;
  }

  # create the subelements for Style
  /*
  e.g.,
    <Style id="118550863">
      <IconStyle>
        <Icon>

<href>http://farm1.static.flickr.com/56/118550863_1b8f5a26aa_s.jpg</href>
        </Icon>
      </IconStyle>
    </Style>
  */

  protected function populate_style($style,$photo) {
    $id = $photo['id'];
    $farmid = $photo['farm'];
    $serverid = $photo['server'];
    $secret = $photo['secret'];
    $square_url =

"http://farm{$farmid}.static.flickr.com/{$serverid}/{$id}_{$secret}_s.jpg";

    $id_attr = $style->setAttributeNode(new DOMAttr('id', $id));
    $iconstyle = $style->appendChild(new DOMElement("IconStyle"));
    $icon = $iconstyle->appendChild (new DOMElement("Icon"));
    $href = $this->attachNewTextNode($icon,"href","",$square_url);

    return $style;
  }

  # converts the response from the Flickr photo search ($rsp),
  # the arguments from the original search ($arg_array),
  # the $path of the script to KML

  public function kml_from_pics($arg_array, $path, $rsp) {

    $xml = simplexml_load_string($rsp);
    $dom = new DOMDocument('1.0', 'UTF-8');
    $kml = $dom->appendChild(new DOMElement('kml'));
    $attr = $kml->setAttributeNode(new DOMAttr('xmlns',
'http://earth.google.com/kml/2.2'));
    $document = $kml->appendChild(new DOMElement('Document'));

    # See http://www.flickr.com/services/api/misc.urls.html
    # Remember http://farm{farm-id}.static.flickr.com/{server-
id}/{id}_{secret}.jpg
    # syntax for URLs

    # parameters for LookAt -- hard-coded in this instance
    $range = 2000;
    $altitude = 0;
    $heading =0;
    $tilt = 0;

    # make the <Style> elements first
    foreach ($xml->photos->photo as $photo) {
      $style = $document->appendChild(new DOMElement('Style'));
      $this->populate_style($style,$photo);
    }

    # now make the <Placemark> elements -- but tuck them under one Folder
    # in the Folder, add URLs for the KML document and how to send
    # the KML document to Google Maps

    $folder = $document->appendChild(new DOMElement('Folder'));
    $folder_name_node = $this->attachNewTextNode($folder,"name","","Flickr
Photos");
    $kml_url =  $path . "?" . form_url_params($arg_array,TRUE);
    $description_string = "Total Number of Photos available: {$xml->
photos['total']}" . "&nbsp;<a href='{$kml_url}'>KML</a>";
    $description_string .= "&nbsp;<a href='" . "http://maps.google.com?q=" .
urlencode($kml_url) .  "'>GMap</a>";
    $folder_description_node = $this->
attachNewTextNode($folder,"description","",$description_string);

    # loop through the photos to convert to a Placemark KML element
    foreach ($xml->photos->photo as $photo) {
      $farmid = $photo['farm'];
      $serverid = $photo['server'];
      $id = $photo['id'];
      $secret = $photo['secret'];
      $owner = $photo['owner'];
      $thumb_url =

"http://farm{$farmid}.static.flickr.com/{$serverid}/{$id}_{$secret}_t.jpg";
      $med_url =

"http://farm{$farmid}.static.flickr.com/{$serverid}/{$id}_{$secret}.jpg";
      $page_url = "http://www.flickr.com/photos/{$owner}/{$id}";
      $image_html= "<a href='{$page_url}'><img src='{$med_url}'></a>";
      $title = $photo['title'];
      $latitude = $photo['latitude'];
      $longitude = $photo['longitude'];
      $placemark = $folder->appendChild(new DOMElement('Placemark'));

      # place the photo title into the <name> KML element
      $name = $this->attachNewTextNode($placemark,"name","",$title);

      # drop the title and thumbnail into description and wrap in CDATA
      # to work around encoding issues
      $description_string = "{$image_html}";
      $description = $placemark->appendChild(new DOMElement('description'));
      $description->appendChild($dom->createCDATASection($description_string));

      $lookat = $placemark->appendChild(new DOMElement('LookAt'));
      $longitude_node = $this-
>attachNewTextNode($lookat,"longitude","",$longitude);
      $latitude_node = $this-
>attachNewTextNode($lookat,"latitude","",$latitude);
      $altitudeNode = $this->attachNewTextNode($lookat,"altitude","",$altitude);
      $altitudeMode =
        $this->attachNewTextNode($lookat,"altitudeMode","","relativeToGround");
      $rangeNode = $this->attachNewTextNode($lookat,"range","",$range);
      $tiltNode = $this->attachNewTextNode($lookat,"tilt","",$tilt);
      $headingNode = $this->attachNewTextNode($lookat,"heading","",$heading);

      $styleurl = $this->attachNewTextNode($placemark, "styleUrl","","#".$id);

      $point = $placemark->appendChild(new DOMElement('Point'));
      $coordinates_string = "{$longitude},{$latitude},{$altitude}";
      $coordinates =
        $this->attachNewTextNode($point,"coordinates","",$coordinates_string);
    }
    return $dom->saveXML();
  }

  # generate a network link based on the user search parameters ($arg_list)
  # and the $path to this script
  public function generate_network_link ($arg_list, $path) {

    # look through the $arg_list but get rid of lat/long and blanks
    unset ($arg_list['lat0']);
    unset ($arg_list['lat1']);
    unset ($arg_list['lon0']);
    unset ($arg_list['lon1']);

    $arg_list['o_format'] = 'kml';  //set to KML
    $url = $path . "?" . form_url_params($arg_list,TRUE);
    $url = xmlentities($url);

    # generate a description string to guide user
    # to reparameterizing the network link
    $arg_list['o_format'] = 'html';
    $url2 = $path . "?" . form_url_params($arg_list,TRUE);
    $description = "<a href='{$url2}'>Search Something Different</a>";
    $nl = <<<EOT
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <NetworkLink>
    <flyToView>0</flyToView>
    <name>Pictures from Flickr</name>
    <description><![CDATA[{$description}]]></description>
    <open>1</open>
    <visibility>1</visibility>
    <Link>
      <href>{$url}</href>
      <viewRefreshMode>onStop</viewRefreshMode>
      <viewRefreshTime>3</viewRefreshTime>
      <viewFormat>lat0=[bboxSouth]&amp;lon0=[bboxWest]&amp;lat1=[bboxNorth]
&amp;lon1=[bboxEast]</viewFormat>
    </Link>
  </NetworkLink>
</kml>
EOT;
   return $nl;
  }
} // flickr_kml

# this class translates what comes in on the URL and from form values
# to parameters to submit to Flickr

class flickr_view {

  # this function filters $_GET passed in as $get by parameters
  # that are in $defaults
  # only parameters named in $defaults are allowed -- and if that value isn't
set in
  # $get, then this function passes back the default value

  public function form_input_to_user_inputs($get,$defaults) {
    $params = array();
    foreach ($defaults as $arg => $default_value) {
      $params[$arg] = isset($get[$arg]) ? $get[$arg] : $default_value;
    }
    return $params;
  }

  # translate the user inputs to the appropriate ones for Flickr.
  # for example -- fold the latitudes and longitude coordinates into bbox
  # get rid of o_format  for Flickr

  public function user_inputs_to_flickr_params($user_inputs) {
    $search_params = $user_inputs;

    $o_format = $user_inputs["o_format"];
    unset ($search_params["o_format"]);

    if (($o_format == "json") || ($o_format == "rest")) {
      $search_params["format"] = $o_format;
    } else {
      $search_params["format"] = "rest";
    }

    #recast the lat and long parameters in bbox

    $bbox = "{$search_params['lon0']},{$search_params['lat0']},
             {$search_params['lon1']},{$search_params['lat1']}";
    $search_params['bbox'] = $bbox;
    unset($search_params['lon0']);
    unset($search_params['lon1']);
    unset($search_params['lat0']);
    unset($search_params['lat1']);

    return $search_params;
  } // user_inputs_to_flickr_params
} //flickr_view

// API key here
$api_key = "[API-KEY]";

# a set of defaults -- center the search around Berkeley by default
# and any geotagged photo in that bounding box.
# BTW, this script needs at least geo in extras.
# min_upload_date corresponds to Jan 1, 1996 (Pacific time)
$default_args = array(
  "user_id" => '',
  "tags" => '',
  "tag_mode" => '',
  "text" => '',
  "min_upload_date" => '820483200',
  "max_upload_date" => '',
  "min_taken_date" => '',
  "max_taken_date" => '',
  "license" => '',
  "sort" => '',
  "privacy_filter" => '',
  "lat0" => 37.81778516606761,
  "lon0" => -122.34374999999999,
  "lat1" => 37.92619056937629,
  "lon1" => -122.17208862304686,
  "accuracy" => '',
  "safe_search" => '',
  "content_type" => '',
  "machine_tags" => '',
  "machine_tag_mode" => '',
  "group_id" => '',
  "place_id" => '',
  "extras" => "geo",
  "per_page" => 10,
  "page" => 1,
  "o_format" => 'html'
);
```

```
# calculate the path to this script as a URL.
$path = "http://" . $_SERVER['SERVER_NAME'] . $_SERVER['PHP_SELF'];

# instantiate the Flickr wrapper and the view object
$fw = new flickrwrapper($api_key);
$fv = new flickr_view();

# get the parameters that have been submitted to it.
$user_inputs = $fv->form_input_to_user_inputs($_GET,$default_args);
$search_params = $fv->user_inputs_to_flickr_params($user_inputs);

# see what the requested format is
$o_format = $user_inputs["o_format"];

# if the user is looking for a network link,
# calculate the Networklink KML and return it
# with the appropriate Content-Type for KML

if ($o_format == 'nl') {
  $fk = new flickr_kml();
  header("Content-Type:application/vnd.google-earth.kml+xml");
  $downloadfile="flickr.kml"; # give a name to appear at the client
  header("Content-disposition: attachment; filename=$downloadfile");
  print $fk->generate_network_link($user_inputs,$path);
  exit();
}

# If the user is looking instead for JSON, REST, HTML, or KML, we query Flickr

$response = $fw->search($search_params);

# If the request is for JSON or REST,
# just pass back the results of the Flickr search
if (($o_format == "json") || ($o_format == "rest")) {
  foreach ($response["headers"] as $header => $val) {
    header("{$header}:{$val}");
  }
  print $response["body"];

  # if the request is for HTML or KML, do the appropriate transformations.

} elseif ($o_format == "html") {

  # now translate to HTML
  $fh = new flickr_html();
  header("Content-Type:text/html");
  print $fh->generate_form($user_inputs, $_SERVER['PHP_SELF']);
  print $fh->html_from_pics($response["body"]);

} elseif ($o_format == "kml") {

  $fk = new flickr_kml();
  header("Content-Type:application/vnd.google-earth.kml+xml");
  $downloadfile="flickr.kml"; # give a name to appear at the client
  header("Content-disposition: attachment; filename=$downloadfile");
  print $fk->kml_from_pics($user_inputs, $path, $response["body"]);

}
?>
```

Summary

The online mapping arena is changing very quickly, and I obviously am not able to cover the details of all these changes. Nonetheless, here's what I believe will be the long-term trends in this area:

* You'll see a migration of many features found in typical full-fledged GIS system—for example, shading of layers—into programmable web applications.
* Not surprisingly, you'll see the platform players (such as Google Maps) incorporate functionality that started off as extensions to the platform into the platform itself. For example, sites such as Mapbuilder.net provided a user interface for building a Google (or Yahoo!) map before Google made it easier to build a Google map via its My Maps functionality. Google's My Maps doesn't exactly duplicate Mapbuilder.net, but it's bound to win a major audience by virtue of its tight integration with Google Maps.
* You will see increased merging in 2D and 3D representations of the globe. As you've already seen in this chapter, Microsoft's Live Search Maps has both 3D and 2D views. You can use KML as a way of moving data between Google Earth and Google Maps. KML is finding support from competitors to Google such Yahoo!'s Flickr and Yahoo! Pipes.
* We're going to see more native support of GPS devices as they become ubiquitous.

Here are some references where you can find more information:

* *Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional* (Apress, 2006)
* *Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional* (Apress, 2006)
* *Google Maps Hacks: Tips & Tools for Geographic Searching and Remixing* (O'Reilly, 2006)
* *Hacking Google Maps and Google Earth* (Wiley, 2006)
* *Web Mapping Illustrated: Using Open Source GIS Toolkits* (O'Reilly, 2005)
* *Mapping Hacks: Tips & Tools for Electronic Cartography* (O'Reilly, 2005)
* Google Mapki[69] (a great wiki and source of information on Google Maps)

69. http://mapki.com/wiki/Main_Page