PART 1

Your First Google Maps

CHAPTER 1

Google Maps and Rails

The last year or so has been an incredibly exciting time for web developers. New tools have come out that make web development easier, more productive, and more fun. A slew of new APIs are available that let you glue together data and services in interesting ways. As developers, we are more empowered with new and interesting technologies than ever before.

This book focuses on one API that has had a particularly profound impact: the Google Maps API. Since you have this book in hand, you're probably already convinced of Google Maps' importance. In case you need a reminder, however, visit Google Maps Mania (`http://googlemapsmania.blogspot.com`) for a view into the sprawling culture of innovation that Google Maps has fostered in the development community. The Google Maps API has spawned a whole class of web-based applications that would have been impossible to create without it.

You're going to use the Google Maps API on a platform that has inspired an equally fervent following: Ruby on Rails. The Rails framework facilitates radical improvements in productivity within its niche: database-backed web applications. Rails is intuitive, powerful, and free. Together, Rails and Google Maps enable you, the developer, to build impressive web-based applications that would have been difficult or impossible two years ago.

Over the course of the coming chapters, you're going to move from simple tasks involving markers and geocoding to more advanced topics, such as how to acquire data, present many data points, and provide a useful and attractive user interface.

There are many reasons why Ruby on Rails is an ideal platform to work with Google Maps. Rails makes it trivial to produce and consume XML, which Google Maps uses extensively. Rails also has built-in support for JSON (JavaScript Object Notation), a concise format for passing structured data from server to browser. Finally, Ruby has some excellent libraries for screen-scraping, which we will employ in later chapters.

We are assuming that you are coming to this book with a certain amount of Rails experience. You probably already have Ruby and Rails installed in a development environment and know how to get an application up and running. If not, don't fear: we list some resources in the sidebar "Just Getting Started with Ruby and Rails?" near the end of this chapter to help you get rolling on Rails. Whatever your current skill level vis-à-vis Rails, this book will get you in the mapping game and tell you everything you need to create killer maps applications. With the power of the Rails framework, the Ruby language, and the Google Maps API, you will command a development toolkit to be reckoned with.

We know you're eager to get started on a map project, but before we dig into the code, we want to show you two simple ways of creating ultraquickie maps: with KML (Keyhole Markup Language) files and through the Wayfaring map site.

Both these approaches are stepping stones; we will use them as easy introductions to the world of Google Maps. In Chapter 2, you will begin digging into code, which will of course lead to much greater flexibility and sophistication in what you can build.

KML: Your First Map

KML is one of the easiest methods to get your own markers and content displayed on a Google map. As you would expect, the ease is at the expense of flexibility, but it's still a great way to get started. In June 2006, Google announced that its official maps site would support the plotting of KML files. You can simply plug a URL into the search box, and Google Maps will show whatever locations are contained in the KML file specified by the URL. We aren't going to go in depth on this, but we've made a quick example to show you how powerful the KML method is, even if it is simple.

■ **Note** The name *Keyhole Markup Language* is a nod to both its XML structure and Google Earth's heritage as an application called Keyhole. Keyhole was acquired by Google in late 2004.

We created a file called `toronto.kml` and placed the contents of Listing 1-1 in it. The paragraph blurbs are borrowed from Wikipedia, and the coordinates were discovered by manually finding the locations on Google Maps.

Listing 1-1. *A Sample KML File*

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.google.com/earth/kml/2">
<Document>
  <name>toronto.kml</name>
  <Placemark>
    <name>CN Tower</name>
    <description> The CN Tower (Canada's National Tower, Canadian National Tower),
    at 553.33 metres (1,815 ft., 5 inches) is the tallest ➥
    freestanding structure on land.
    It is located in the city of Toronto, Ontario, Canada, and is considered the
    signature icon of the city. The CN Tower attracts close to two million visitors
    annually.

    http://en.wikipedia.org/wiki/CN_Tower</description>
    <Point>
      <coordinates>-79.386864,43.642426</coordinates>
    </Point>
  </Placemark>
</Document>
</kml>
```

In the actual file (located at `http://book.earthcode.com/kml/toronto.kml`), we included two more `Placemark` elements that point to other well-known buildings in Toronto. To view this on Google Maps, paste the previous URL into the Google Maps search field. Alternatively, you can just visit the following link: `http://maps.google.com/maps?f=q&hl=en&q=http://book.earthcode.com/kml/toronto.kml`.

Figure 1-1 shows what it looks like.

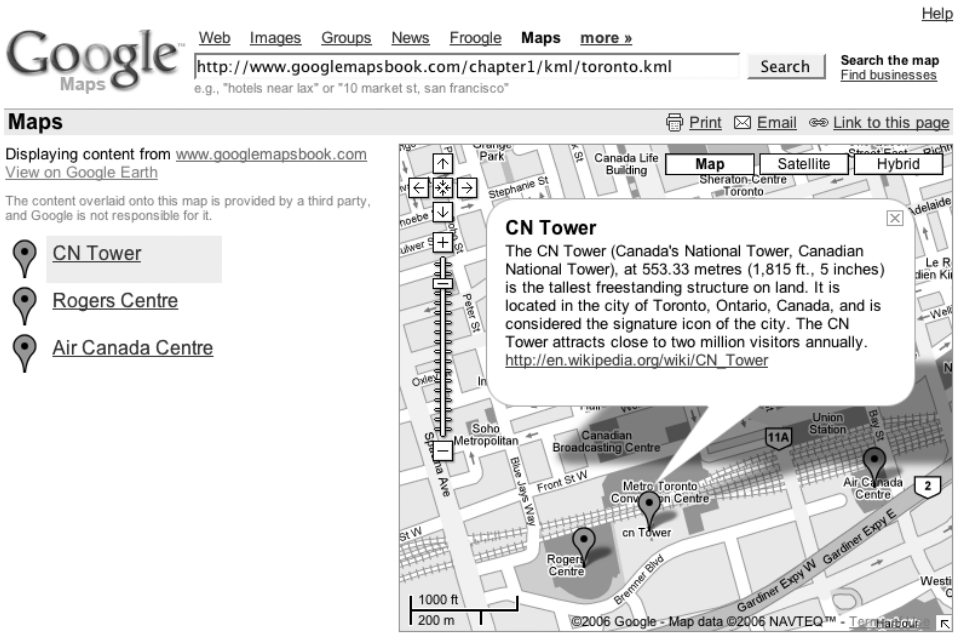

Figure 1-1. *A custom KML data file being displayed at maps.google.com*

Now, is that a quick result or what? Indeed, if all you need to do is show a bunch of locations, it's possible that a KML file will serve your purpose. If you're trying to link to your favorite fishing spots, you could make up a KML file, host it somewhere for free, and be finished.

But that wouldn't be any fun, would it? After all, as cool as the KML mapping is, it doesn't actually offer any interactivity to the user. In fact, most of the examples you'll work through in Chapter 2 are just replicating the functionality that Google provides here out of the box. But once you get to Chapter 3, you'll start to see things that you can do *only* when you harness the full power of the Google Maps API.

Before moving on, though, we'll take a look at one other way of getting a map online quickly.

Wayfaring: Your Second Map

A number of services out there let you publish free maps of quick plotted-by-hand data. One of these, which we'll demonstrate here, is Wayfaring, shown in Figure 1-2. Wayfaring has received attention and praise for its classy design and community features (such as commenting and shared locations). Wayfaring is also built using Ruby on Rails.

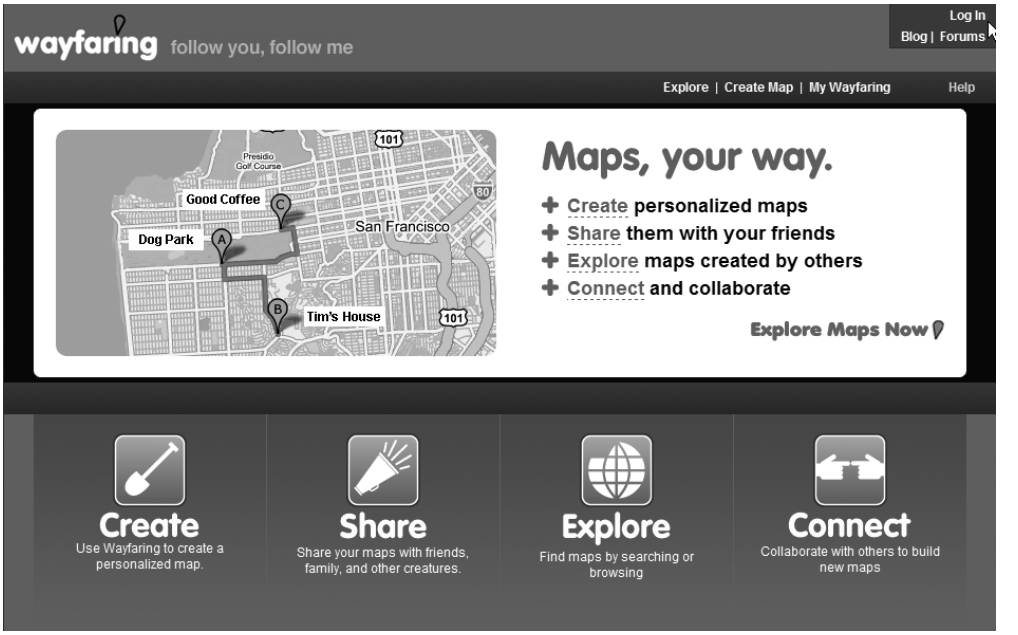

Figure 1-2. *Wayfaring home page*

Wayfaring is a mapping service that uses the Google Maps API and allows users to quickly create maps of anything they like. For example, some people make maps of their vacations; others have identified interesting aspects of their hometown or city. We'll walk you through making a quick map of an imaginary trip to the Googleplex in Mountain View, California.

Point your browser at `http://www.wayfaring.com` and follow the links to sign up for an account (clicking Log In will bring up the option to create a new account). Once you've created and activated your account, you can begin building your map by clicking the Create Map link in the upper right.

Adding the First Point

Let's start by adding the home airport for our imaginary journey. We're going to use Lester B. Pearson International Airport in Toronto, Ontario, Canada, but you could use the airport closest to you. Since Pearson is an international location (outside the United States), you need to drag and zoom the map view until you find it. If you're in the United States, you can use the nifty Jump To feature to search by text string. Figure 1-3 shows Pearson nicely centered and zoomed.

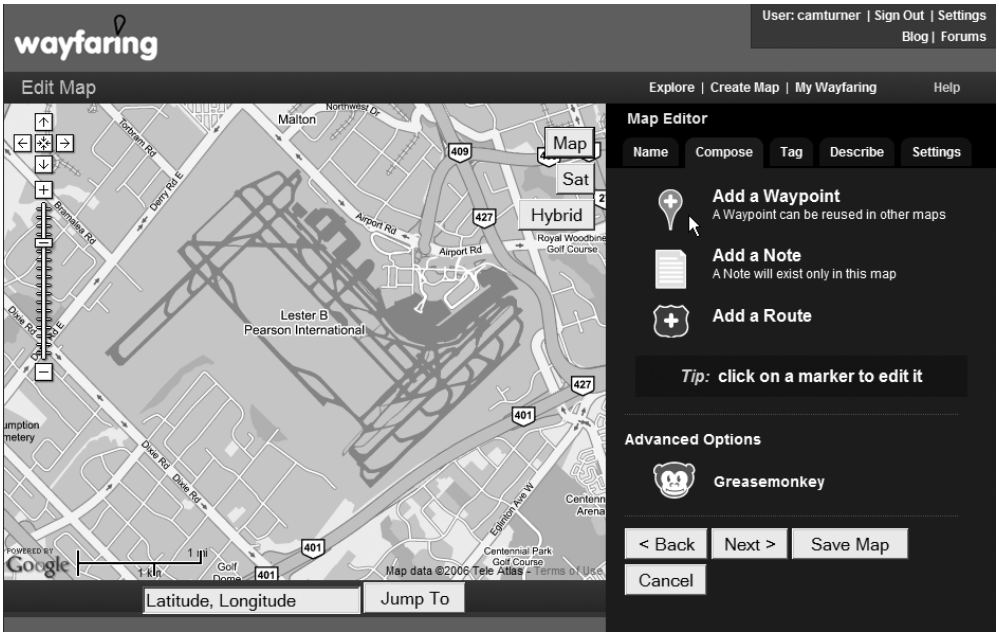

Figure 1-3. *Lester B. Pearson International Airport, Toronto, Ontario*

Once you've found your airport, you can click Next and name the map. Click Next again (after naming your map), and you should be back at the main Map Editor screen.

Select Add a waypoint from the list of options on the right. You'll be prompted to name the waypoint. We'll call ours *Lester B. Pearson International Airport*. However, as you type, you'll find that Wayfaring suggests this exact name. This means that someone else on some other map has already used this waypoint, and the system is giving you a choice of using their point or making one of your own. It's a safe bet that most of the airports you could fly from are already in Wayfaring, so feel free to use the suggested one if you would like. For the sake of the learning experience, let's quickly make our own. Click Next to continue.

The next two screens ask you to tag and describe this point in order to make your map more searchable for other members. We'll add the tags "airport Toronto Ontario Canada" and give it a simple description. Finally, click Done to commit the point to the map, which returns you to the Map Editor screen.

Adding the Flight Route

The next element you're going to add to your map is a *route*. A route is a line made up of as many points as you like. We'll use two routes in this example. The first will be a straight line between the two airports to get a rough idea of the distance the plane will have to travel to get us to Google's headquarters. The second will be used to plot the driving path we intend to take between the San Francisco airport and the Googleplex.

To begin, click Add a Route, name the route (something like *airplane trip*), and then click your airport. A small white dot appears on the place you click. This is the first point on your line. Now zoom out, scroll over to California, and zoom in on San Francisco. The airport is on the west side of the bay. Click the airport here, too. As you can see in Figure 1-4, a second white dot appears on the airport, and a blue line connects the two points. You can see the distance of the flight on the right side of the screen, underneath the route label. Wow, the flight seems to have been more than 2,000 miles! If you make a mistake and accidentally click on the map a few extra times (thereby creating extraneous midway points) in the process of getting to San Francisco, you can use the Undo Last option. Otherwise, click Save.

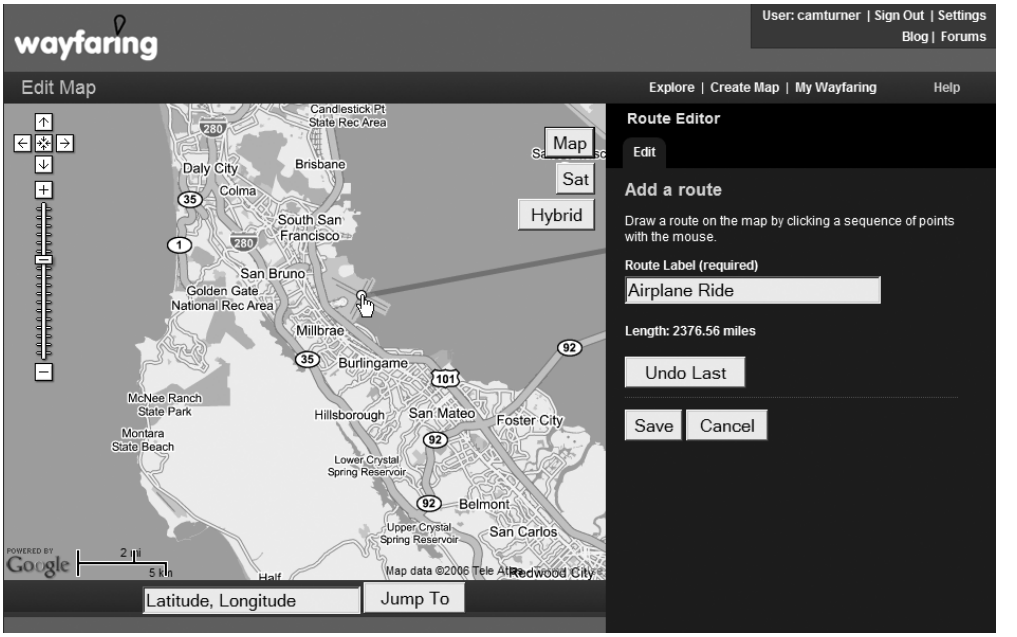

Figure 1-4. *Your flight landing at San Francisco International Airport*

Adding the Destination Point

Now that you're in San Francisco, let's figure out how to get to the Googleplex directly. Click Add a Waypoint. Your destination is Google, so the new point is called *The Googleplex*. Use the address box feature to jump directly to 1600 Amphitheatre Parkway, Mountain View, California, 94043. Wayfaring is able to determine latitude and longitude from an address via a process called *geocoding*, which you'll see a lot more of in Chapter 4.

To confirm you're in the right place, click the Sat button on the top-right corner of the map to switch it over to satellite mode. You should see something resembling Figure 1-5.

Figure 1-5. *The Googleplex*

Adding a Driving Route

Next, let's figure out how far the drive is. Routes don't really have a starting and ending point in Wayfaring from a visual point of view, so you can start your route from the Googleplex and work your way backward. Switch back into Map mode or Hybrid mode so you can see the roads more clearly. From the Map Editor screen, select Add a Route and click the point you just added. Use 10 to 20 dots to carefully trace the trip from Mountain View back up the Bayshore Freeway (U.S. Highway 101) to the airport. You'll end up with about 23 miles of driving, as shown in Figure 1-6.

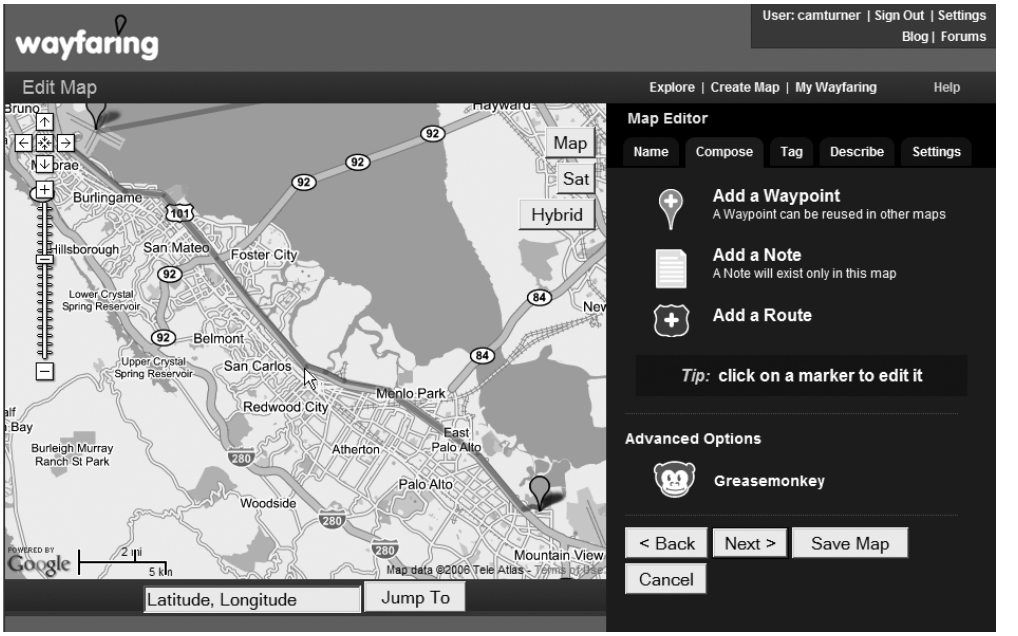

Figure 1-6. *The drive down the Bayshore Freeway to the Googleplex*

That's it. You can use the same principles to make an annotated map of your vacation or calculate how far you're going to travel; and best of all, it's a snap to share it. To see this map live, visit `http://www.wayfaring.com/maps/show/17131`.

Got Rails?

Of course, since this *is* a programming book, you're probably eager to dig into the code and make something really unique. Wayfaring may be nice, but it doesn't give you the flexibility of a programmatic approach. Looking forward to more programmatic interaction with the API, let's discuss Ruby on Rails, the server-side framework of choice.

As we mentioned at the beginning of this chapter, this book presumes you have some experience with Ruby and Rails already—at least enough to get a basic Rails application up and running. In particular, you should

- Have Ruby (1.8.4 or later) and Rails (1.1.5 or later) installed on your development machine.
- Know how to use the `rails` command to create an application skeleton and be comfortable with the model/view/controller layout of a Rails application.
- Know how to start a WEBrick, Mongrel, or other server to view your pages in a browser.
- Have a local database server. We use MySQL throughout this book, but you should be able to adapt to the DB engine of your choice.

- Have a development environment that you're comfortable with—for example, TextMate, RadRails, or Vim.

The following are things we *don't* expect you to know (or have) coming into this book:

- You don't need to know the details of the `prototype.js` JavaScript library. A lot of developers are confused by Prototype, probably due to the dearth of documentation. We've made a conscious decision in this book to not rely heavily on `prototype.js`, although we will be utilizing it from time to time.
- You don't need to know RJS. RJS is a Rails template type (like RHTML) and is a convenient way to build rich functionality on web pages with a minimum of JavaScript coding. Since the Google Maps API is implemented in JavaScript, we'll rely primarily on hand-coded JavaScript functions for this book.
- You don't need to be an expert in JavaScript programming. Yes, the Google Maps API is implemented in JavaScript, but you'll be ramping up on JavaScript techniques and principles as you go.
- You don't necessarily need to have a production web space for your Rails application. Of course, you'll want a production server so you can expose your killer app to the world, but you can learn everything in this book using your local development server.

JUST GETTING STARTED WITH RUBY AND RAILS?

If you're just getting started with Ruby and Rails and decided mapping applications are a fun way to learn, you'll probably want some additional resources to help you get up to speed. Ruby on Rails has a steeper learning curve than PHP or ColdFusion, so you will benefit from these resources to get you started out right:

- *Beginning Ruby: From Novice to Professional* by Peter Cooper (`http://www.apress.com/book/bookDisplay.html?bID=10244`). We recognize that many developers' first exposure to Ruby is through the Rails web framework. If this is the case for you, I highly recommend learning more about the wonderful Ruby language before diving headfirst into Rails.
- *Beginning Ruby on Rails: From Novice to Professional* by Cloves Carneiro Jr. and Jeffrey Allan Hardy (`http://www.apress.com/book/bookDisplay.html?bID=10124`).

What's Next?

We hope you're eager to learn how to build your own maps-based applications from the ground up using Rails. By the end of Part 1 of this book, you'll have the skills to do everything you've just done on Wayfaring (except the polylines and distances, which are covered in Chapter 10) using JavaScript and XHTML. By the book's conclusion, you'll have learned most of the concepts needed to build your own Wayfaring clone.

So what exactly is to come? This book is divided into three parts and two appendixes. Part 1 goes through Chapter 4 and deals with the basics that a hobbyist would need to get started.

You'll make a map, add some custom pins, and geocode a set of data using freely available services. Part 2 (Chapters 5 through 8) gets into more map development topics, such as building a usable interface, dealing with extremely large groups of points, and finding sources of raw information you may need to make your professional map ideas a reality. Part 3 (Chapters 9 through 11) dives into advanced topics: building custom map overlays, such as your own info window and tool tip; creating your own map tiles and projections; using the spherical equations necessary to calculate surface areas on the earth; and building your own geocoder from scratch. Finally, one appendix provides a reference guide to the Google Maps version 2 API, and another points to a few places where you can find neat data for extending the examples here and to inspire your own projects.