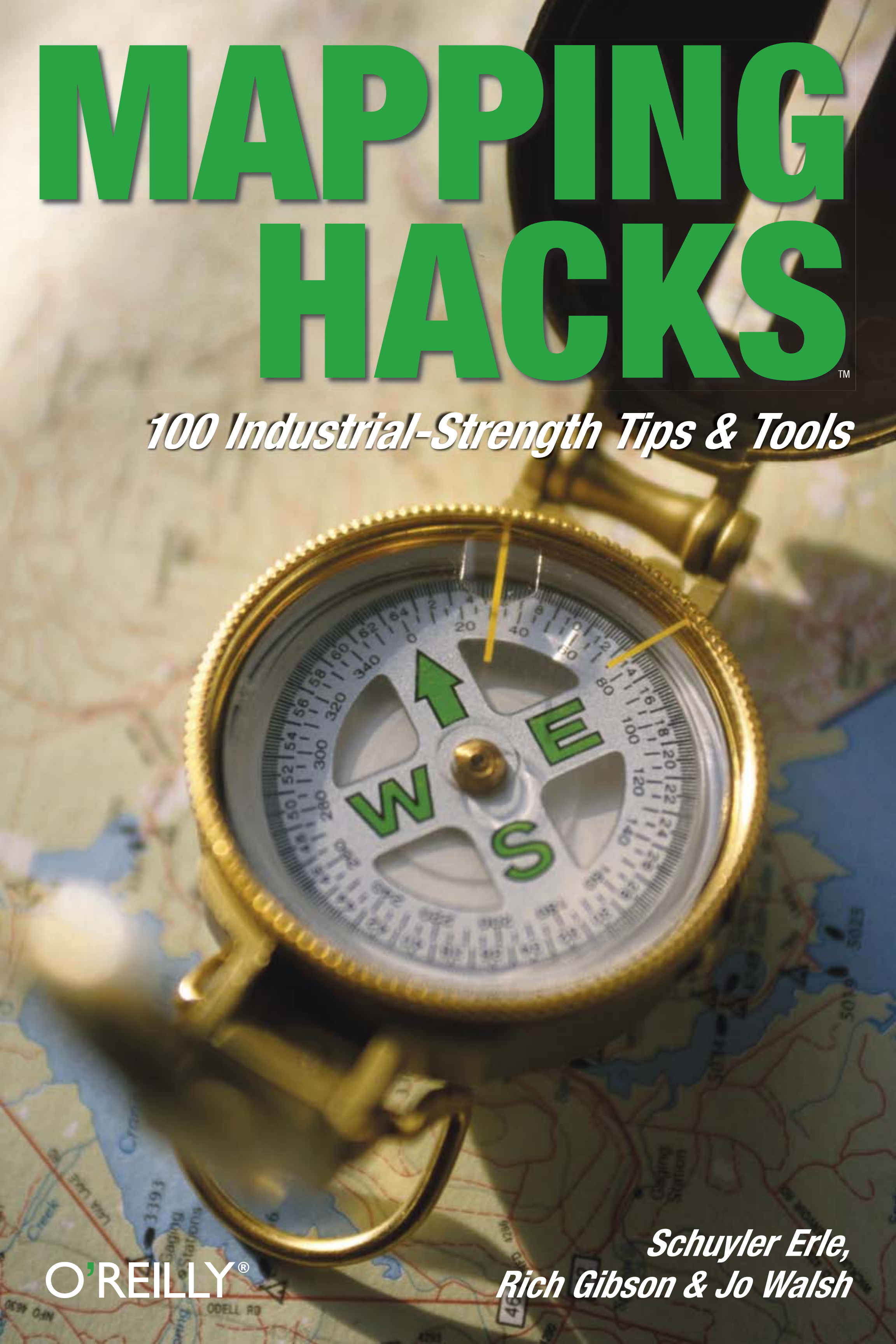
# MAPPING HACKS™

100 Industrial-Strength Tips & Tools



Schuyler Erle,
Rich Gibson & Jo Walsh

O'REILLY®

## HACK #97 Set Up an OpenGuide for Your Hometown

OpenGuides are great, but what if there are none near you?

A *wiki* is a form of web site that is growing massively in popularity. Readers are allowed, even encouraged, to edit and add to the pages on a wiki site. Wikis are a great way to implement a collaboratively written web site. The wiki encyclopedia, Wikipedia **[Hack #45]** (at *http://www.wikipedia.org/*) is an impressive and addictive example.

A guide to your hometown is very well suited to the wiki format, too; real people on the ground can add and annotate their favorite places. Locals can share knowledge that goes deeper than "find me my nearest branch of Starbucks or WHSmith," and it will make for a more interesting read for travelers than guides to tourist hotspots and pricey eateries published by hotel consortiums.

OpenGuides is a wiki software package enhanced with geospatial attributes. Each node or page can be given a postal code, spatial grid coordinates, venue opening times, and addresses. OpenGuides was originally written to provide a spatial-wiki backend for the Open Guide to London, once known as Grubstreet, which was started so that real geeks could tell each other where to find real ale. Figure 9-9 shows the front page of the Open Guide to London.

### Installing OpenGuides from CPAN

OpenGuides is a Perl/CGI-based application and needs a web server to run it. OpenGuides is available from Perl's archive site, CPAN (*http://cpan.org/*). It's based on an extensible wiki framework that allows you to add any kind of structured metadata to your wiki pages. You can install modules and applications from CPAN automatically, using the CPAN shell; this comes with your installation of *perl* by default.

First let's install `CGI::Wiki` using the CPAN shell. As the superuser, type:

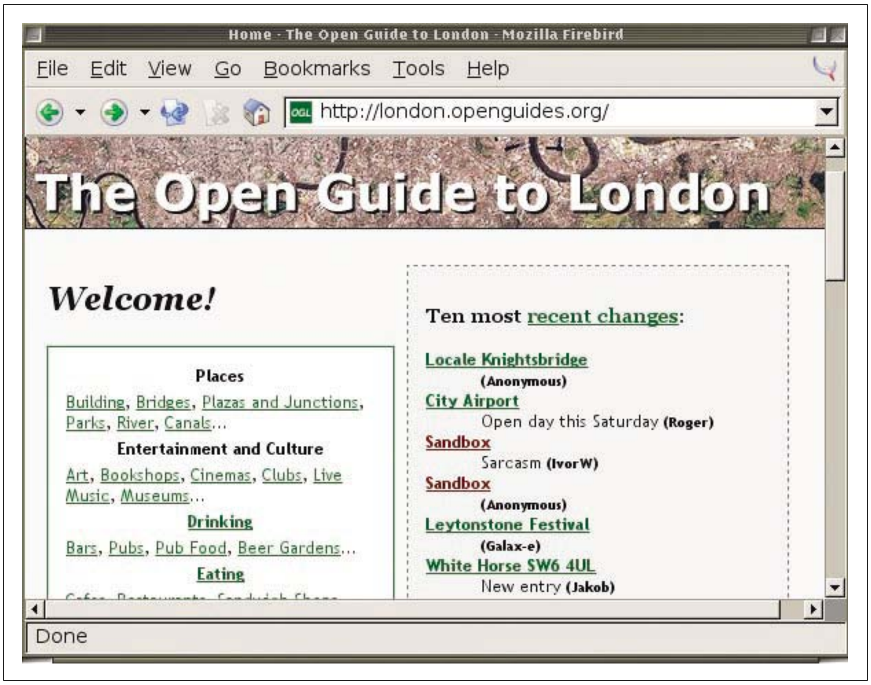
```
# perl -MCPAN -e shell

cpan shell -- CPAN exploration and modules installation (v1.7601)
ReadLine support enabled

cpan> install CGI::Wiki
```

OpenGuides needs the `Config::Tiny` module in order to be able to configure itself while it installs. Also, we'll be using the lightweight SQLite backend to store the wiki pages, so let's install that, too:

```
cpan> install Config::Tiny
cpan> install DBD::SQLite
```


Home - The Open Guide to London - Mozilla Firebird
File Edit View Go Bookmarks Tools Help
http://london.openguides.org/
The Open Guide to London
Done

Welcome!

Places

Building, Bridges, Plazas and Junctions, Parks, River, Canals...

Entertainment and Culture

Art, Bookshops, Cinemas, Clubs, Live Music, Museums...

Drinking

Bars, Pubs, Pub Food, Beer Gardens...

Eating

Ten most recent changes:

Locale Knightsbridge
(Anonymous)
City Airport
Open day this Saturday (Roger)
Sandbox
Sarcasm (IvorW)
Sandbox
(Anonymous)
Leytonstone Festival
(Galax-e)
White Horse SW6 4UL
New entry (Jakob)

*Figure 9-9. The Open Guide to London*

```
cpan> install OpenGuides
```

## Configuring Your OpenGuide

Once the OpenGuides code is downloaded onto your system, the configuration process begins: a simple walkthrough, answering questions with sensible defaults. If everything succeeds, you should see the following prompts:

```
Continue with install? [y] y
Skip OpenGuides configuration? [n] n
```

As this is the first time we've installed OpenGuides on this system, we need to create a configuration file, so don't stop here! The first task is to choose a backend database to use to store the pages; there is support for MySQL, PostgreSQL, and SQLite:

```
What type of database do you want the site to run on? SQLite
```

For this database, we're going to use SQLite, a tiny SQL store that uses a single binary file for the backend and doesn't require a database username and password. You need to make sure that the web server has permissions to write to the SQLite database and has write permissions on the directory that contains it.

Once you've set up the database, the configurer will ask you a range of questions about where you want the OpenGuides scripts to be installed, what address they map to on the Web, and various meta-information about your new wiki—its name, city, country, and language. You can change these options afterward by editing the file *wiki.conf*, which installs into the same directory as the CGI scripts.

You need to make sure that the user or group your web server runs as has permission to write to files in the directory where you store the indexes. If you're running the Apache HTTP server, this is likely to be *apache*, *httpd*, *www-data*; check your process table by looking for the `httpd` process:

```
$ ps -ef | grep apache
apache  5031  5019  0 19:42 pts/4    00:00:00 /usr/sbin/apache-perl
$ groups apache
apache : apache
```

Then set the permissions so that the web server can write to the directory, first adding its group and then making the directory group writable:

```
$ cd ~openguides/indexes
$ chgrp apache .
$ chmod g+w .
```

## Configuring Your Web Server

Now you need to tell your web server where you've installed OpenGuides. If it's in a regular *cgi-bin* (i.e., a directory from which the web server already allows execution of programs called from the Web), you shouldn't need to do anything. Otherwise you have to tell your web server where the executable OpenGuides scripts can be found. The following lines are a sample Apache web server configuration for OpenGuides, which you would typically add inside a `<VirtualHost>` section:

```
ScriptAlias /openguides/ /home/www/openguides/
<Directory /home/www/openguides>
    AllowOverride None
    Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
    Order allow,deny
    Allow from all
</Directory>
```

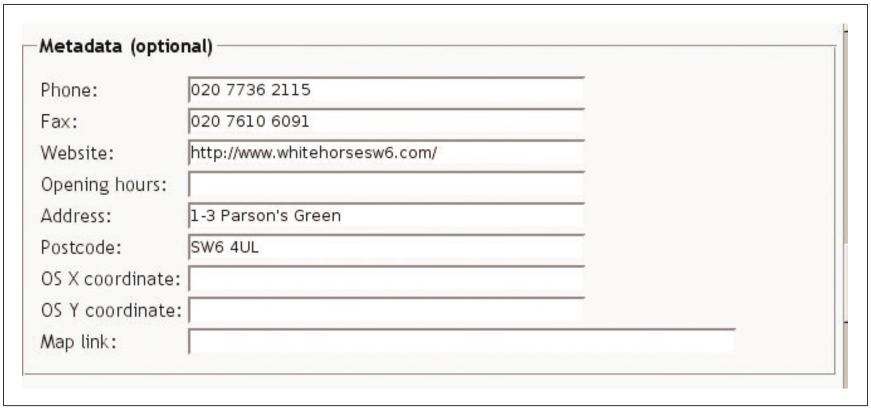
Now you should be able to view your OpenGuide on the Web! Open a web browser and visit the URL that maps to your *cgi-bin*, or wherever you just configured your OpenGuide to live.

## Customizing Your OpenGuide

You should now see the front page of your new site; it looks very bare. Click the "Edit this page" link to edit the front page, or select the "Create a new page" link. Then start documenting your favorite interesting pages!

When you edit or create a new page in OpenGuides, you'll see an edit form that will be familiar to you if you've ever used a wiki. The top box in the form is for a description, review, or commentary about the place in question.

There are extra options for place metadata, which make OpenGuides quite different from an ordinary wiki. Figure 9-10 shows some of these options. Each page can go in one or several Categories, and also in one or more Locales. A Locale is a neighborhood or other local area. OpenGuides allows people to define what they think belongs in particular "areas" on a freeform basis.


**Metadata (optional)**

| | |
|---|---|
| Phone: | 020 7736 2115 |
| Fax: | 020 7610 6091 |
| Website: | http://www.whitehorsesw6.com/ |
| Opening hours: | |
| Address: | 1-3 Parson's Green |
| Postcode: | SW6 4UL |
| OS X coordinate: | |
| OS Y coordinate: | |
| Map link: | |

*Figure 9-10. Editing page metadata in OpenGuides*

There's also a lot of optional metadata for each page: contact details, opening hours, web links, etc. There's also space for interesting spatial metadata: a street address and postal code, a link to a map, and coordinates.

OpenGuides comes from the UK. It was originally developed to power the flagship London guide, *http://london.openguides.org/*. If you're based in the UK, there are options to add an Ordnance Survey grid reference as *os_x* and *os_y* coordinate fields. The British National Grid is a Transverse Mercator projection of the UK, with the first digit of the coordinates omitted, because it is invariant within the coverage area.

Adding latitude and longitude—available, for example, from the efficient *http://www.streetmap.co.uk/*—allows OpenGuides to provide a "search for nearby places" option on each page, which shows geocoded pages of nearby places. OpenGuides also makes a decent attempt to geocode postcodes. Some international sites do this differently; the Open Guide to San Francisco (*http://sf.openguides.org*) has been adapted to look up street addresses using the free geocoder service at *http://geocoder.us/*. **[Hack #79]** has more information on how this is done.

Think about what kinds of categories of things, and what kind of neighborhoods, you want your OpenGuide to talk about. Inevitably people will create all manner of weird ad hoc categories on their own (this is a wiki after all), but a bit of "wiki gardening" will give people some direction.

OpenGuides has an active and helpful developer community. You can find their contact details on *http://openguides.org/*. They are prepared to host individual OpenGuides for people who don't have access to a web server and will also delegate DNS subdomains under *openguides.org* to new OpenGuides upon request.

## Make Maps of Your OpenGuide

As your OpenGuide expands, you'll probably want to see a map illustrating the scope of your virtual terrain. Luckily, the OpenGuides software gives you a good place to start; each page in the Guide has an RDF/XML feed that covers its spatial metadata in machine-comprehensible form. "Model Interactive Spaces" **[Hack #95]** outlines the basics of using the Resource Description Framework to annotate space. Right now, we can jump straight in with the help, as usual, of a quick Perl script.

The original and finest site based on the OpenGuides software is the Open Guide to London (*http://london.openguides.org/*). Each Open Guide has an RDF/XML index of all its pages, which you can read by appending `index.cgi?action=index;format=rdf` to the main site URL. This index page gives us pointers to all the different pages in any Open Guide: Categories, Locales (a Unix-ish term for a neighborhood or area), and each place listed in the Guide itself. The index doesn't show the latitude and longitude of each place, though; for that, you have to look at the RDF version of each page.

For example, if the HTML version of a wiki page is at *http://london.openguides.org/index.cgi?Devonshire_Arms,_NW1_8NL*, then there will be an RDF representation of it at *http://london.openguides.org/index.cgi?id=Devonshire_Arms,_NW1_8NL&format=rdf*. The metadata looks something like this:

```
<geo:SpatialThing rdf:ID="obj" dc:title="Devonshire Arms, NW1 8NL">
 <!-- categories -->
  <dc:subject>Camden</dc:subject>
  <dc:subject>Goth</dc:subject>
  <dc:subject>Pubs</dc:subject>
 <!-- address and geospatial data -->
  <city>London</city>    <postalCode>NW1 8NL</postalCode>
  <country>United Kingdom</country>
  <wn:Neighborhood>Camden</wn:Neighborhood>
  <geo:lat>51.539948</geo:lat>
  <geo:long>-0.140826</geo:long>
```

```
    <!-- other attributes -->
  </geo:SpatialThing>
```

The following chunk of Perl illustrates how to collect and parse spatial data from an OpenGuide. We use the `RDF::Simple` module to parse the index page and extract everything in it that is a `geo:SpatialThing` and therefore likely to have coordinates. For each space we find, we download the RDF version of its page. If there are RDF "triples" for latitude and longitude in that page, we store the lat/long pair, along with the URL pointing to the place, in a list of points.

As usual, there are caveats—that's why this is a "hacks" book and not a "perfect solutions" book. The index page has a couple of XML-toxic characters in it, probably created by a Windows user cutting and pasting a new page name into the wiki; we can get rid of these with a regular expression before passing the document to the RDF parser. There are also a few points that have a latitude and longitude of 0,0, coordinates that are far outside London, a result of people typing coordinates in by hand and making little mistakes.

The original *http://geourl.org* service had a lot of user-input problems, resulting in the fabled "Ghost Blogs of Tibet." We strongly advise that if you're developing a mapping application where people have to type in coordinates, the application should show those coordinates plotted on a map right away, providing visual feedback. See "Plot Arbitrary Points on a World Map" **[Hack #29]** for a simple way to do this.

Finally, some of the points are in old-fashioned degrees-minutes-seconds format, rather than newfangled decimal format. Catch these with a regular expression, and it's simple to convert between the two:

```
#!/usr/bin/perl

use strict;
use RDF::Simple::Parser;
use LWP::Simple;

my $parser = RDF::Simple::Parser->new;
my $rdf = get('http://london.openguides.org/index.
cgi?action=index&format=rdf');
$rdf =~ tr/\240/\040/; # XML-toxic characters
my @triples = $parser->parse_rdf($rdf);

# find all triples where the subject is of rdf:type geo:SpatialThing
my @spaces = grep {$_->[2] =~ /SpatialThing/} @triples;
my @points; # an empty list to hold our points
```

```
foreach my $s (@spaces) {
    my $url = $s->[0]; # get the subject url of the triple
    warn("reading $url");
    my @t = $parser->parse_uri($url);
    my ($lat) = grep {$_->[1] =~ /wgs84_pos\#lat/} @t;
    next unless $lat; # don't bother if we can't find a latitude
    next if $lat =~ m/0d/ or $lat !~ m/51/; # we know these are wrong

    my ($lon) = grep {$_->[1] =~ /wgs84_pos\#long/} @t;

    if ($lat =~ m/\d+d \d+m/) {
        # this point is in DMS, not decimal, format
        foreach ($lat,$lon) {
           my ($direction,$d,$m,$s) = $_ =~ /(-?)([\d]+)d (\d+)m ([\d\.]+)s/;
           $_ = $d + $m/60 + $s/3600;
           $_ = 0-$_ if $direction;
        }
    }
    push @points, { lat => $lat, long => $lon, url => $url};
}
```

## Plotting Places over Open Space

Now that we have a list of points, we're ready to roll! What we can do with them, though, depends on where in the world we are. In London, bereft of copyright-free, public domain base maps, the first visualization of OpenGuides was just a plot of points on a blank white background.

The SVG::Plot module, available from CPAN, was written to do this. The module accepts a set of points, each one with an optional URL to turn into a hyperlink, and generates a plot of the points in SVG. Optionally, you can split the points into groups and style each set of points differently—for example, color pubs red and restaurants green, or color the points in each locale differently.

SVG::Plot is indifferent to spatial projections; it will quite cheerfully plot any set of *(x, y)* coordinates to the best of its ability, and work out a height and width for the image, if you don't specify one, from the extents implied by the top left and bottom right points.

If you just hand SVG::Plot a list of latitude-longitude points, you'll quickly run into problems. First, the points will be *unprojected*. As you head further north on Spaceship Earth, the distortion of longitude against latitude becomes greater. At 60 degrees north, degrees of longitude are half the width in meters than they are at the equator, so you'll wind up with a very long, thin, wrong-looking map. Second, SVG only displays at a resolution of one pixel and rounds up or down all decimal fractions of a point. If you try to plot latitude and longitude values over the relatively small area of a city, you'll wind up with one big blob of points layered on top of each other!

For countries that are much taller than they are wide, such as the main landmass of the UK, UTM is often the best-looking projection. (The UK Ordnance Survey's grid system is based on a Transverse Mercator projection.) To convert coordinates to UTM, you have to know which of 60 zones you're in; with the map at *http://www.dmap.co.uk/utmworld.htm*, you can find this out in seconds. For more on UTM conversion, see "Work with Different Coordinate Systems" **[Hack #26]**

A quick and sophisticated solution is to use the magical *PROJ.4* library to convert your set of lat/long points into an *(x, y)* Cartesian projection. `Geo::Proj4` is another of the Perl libraries created during the writing of this book to accomplish this task. The following scrap of Perl continues on from the previous script; we already have the list of points available:

```
use Geo::Proj4;
use SVG::Plot;

my @plot;
my $proj = Geo::Proj4->new( proj => "utm", zone => 30 );
foreach (@points) {
    my ($x, $y) = $proj->forward($_->{lat},$_->{long});

    push @plot, [$x,$y,$_->{url}];
}
my $plot = SVG::Plot->new(points => \@plot,scale => 0.01);
print $plot->plot;
```

The result is shown in Figure 9-11.

## Plotting Places over Online Map Services

In the United States, due to the excellent free data provided by the Census Bureau and Geological Survey, there are many better options for do-it-yourself mapmaking. The Tiger Map Server **[Hack #14]**, provided and maintained for free by the USGS, allows you to attach the URL of a file listing points in a simple "lat,long,name" format and get back a flat map with your points on it. To do this, just add the following set of lines to the point-collection Perl script shown earlier:

```
open(FILE, ">points.txt") or die  $!;  # write  to this file, if we're
allowed
foreach  my $p (@points) {
    print FILE $p->{lat}.' ,'.$p->{long}.','.$p->{url}."\n";
}
close FILE;
```

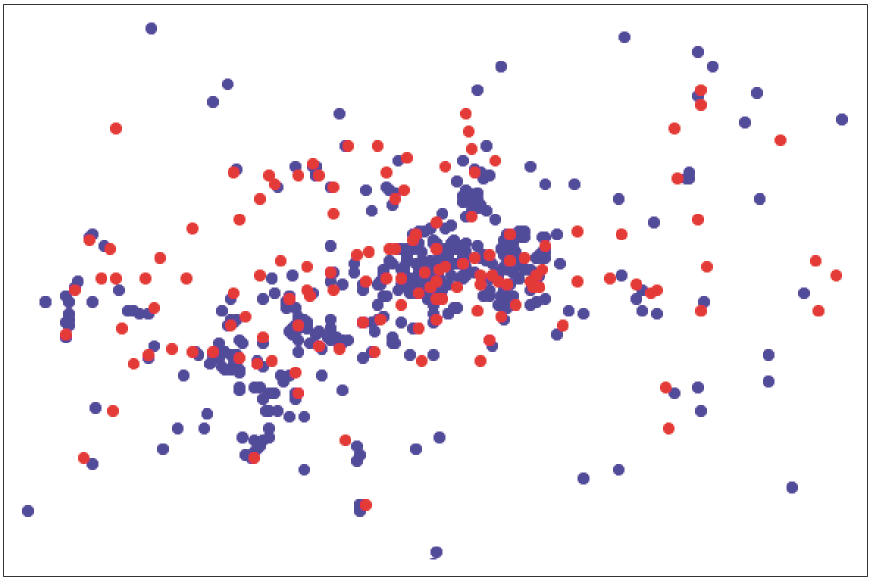
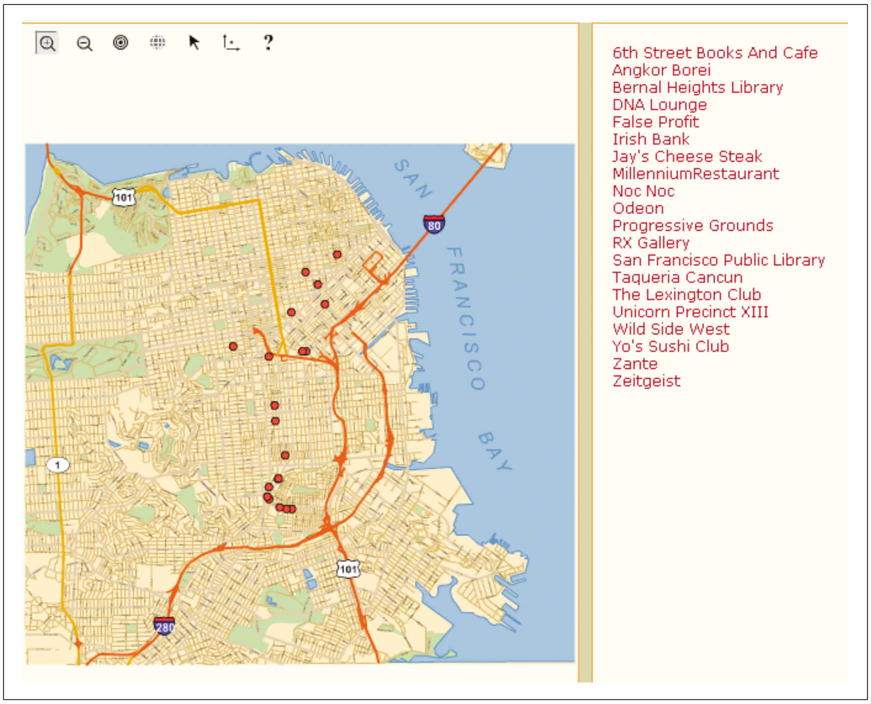


*Figure 9-11. SVG point plot of London Open Guide*

TIGER Map Server will produce a nice static visualization of your point plot, allowing you to use different markers as point symbols, but without much flexibility or interactivity. For a more dynamic map, the services provided free for nonprofit use at *http://mapbureau.com/* are a good way to get started quickly. Map Bureau provides beautiful base maps, created by Donalda Speight, for large U.S. coastal cities. We used the San Francisco base map at Map Bureau, shown in Figure 9-12, to visualize the experimental Open Guide to San Francisco.

You can POST it a URL of a document containing RDF/XML and use one of its base maps to plot the referenced points. (See "Map Health Code Violations with RDFMapper" **[Hack #21]**.) You can also supply your own base map, provided you know the projection the map is in, its origin (i.e., the coordinates for its north-west-most corner), and its extent (i.e., the geographical distance represented by its height and width). To point RDFMapper to your own custom base map, you need to create a file resembling, for example, *http://mapbureau.com/basemaps/sanfrancisco.0.xml*. See *http://mapbureau.com/rdfmapper/* for comprehensive documentation on creating base map files.



6th Street Books And Cafe
Angkor Borei
Bernal Heights Library
DNA Lounge
False Profit
Irish Bank
Jay's Cheese Steak
MillenniumRestaurant
Noc Noc
Odeon
Progressive Grounds
RX Gallery
San Francisco Public Library
Taqueria Cancun
The Lexington Club
Unicorn Precinct XIII
Wild Side West
Yo's Sushi Club
Zante
Zeitgeist

*Figure 9-12. Map Bureau map of San Francisco Open Guide*