Perl script fights parking tickets

Parking Aid

Volunteers mapping neighborhood streets for the free OpenStreetMap project really do pay attention to detail, such as neigborhood parking zones. This month's Perl scripts query the data. *By Michael Schilli*

If you take a look at the current OpenStreetMap of downtown San Francisco (Figure 1), or any other major city of the world, you will see astonishingly detailed map data that has been meticulously collected and updated by volunteers using portable GPS devices. The project doesn't just plot streets and assign names; it also locates bus stops, train tracks, and cycle paths. Thanks to its crowd-source approach, OpenStreetMap (OSM) is frequently more up to date than commercial providers like Google Maps when it comes to stores and restaurants that change their names frequently.

Free Data Instead of Google Maps

One advantage compared with commercial providers is that you are free to download the map data and use it pretty much any way you like. You can click on the website's *Export* button or use an API to access `api.openstreetmap.org` with a program to obtain the map's underlying XML data. This opens up the door for creative tinkering.

The data model is extremely simple: So-called "nodes" designate waypoints, defined by their geographical longitude and latitude in the physical world. Two nodes may be connected by a so-called `way` in the data model, mapping the course of a street, which may be made up by one or more `way` objects.

Figure 2 shows the XML representation of the OSM data for San Francisco's Sutter Street, close to the city's highest skyscrapers. A single click on the *Export* tab in the map view displays the dialog shown in Figure 3, and after you select the *XML* option and press the *Submit* button, your web browser will download an XML file of the

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

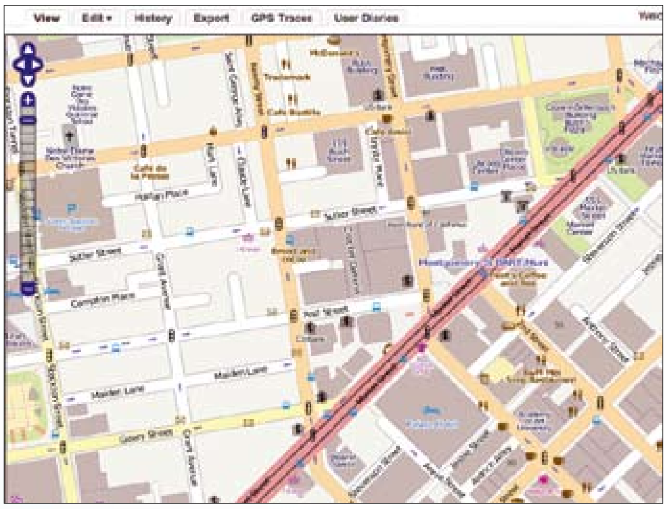


Figure 1: The OpenStreetMap project displaying a free map of downtown San Francisco.

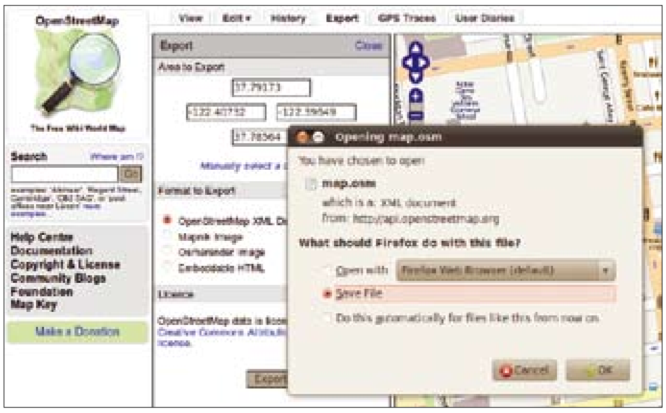


Figure 3: The XML data that drive the map are available for public download and use.

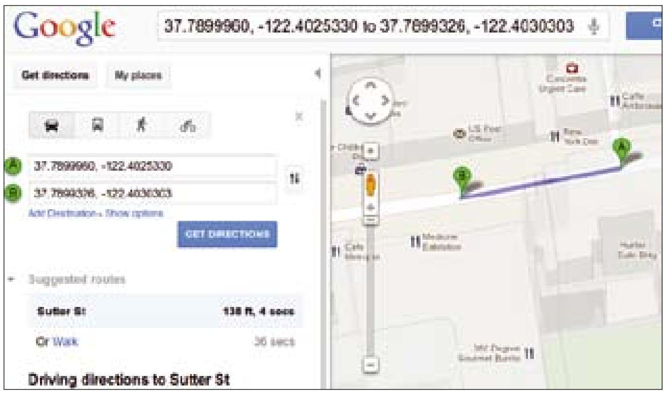
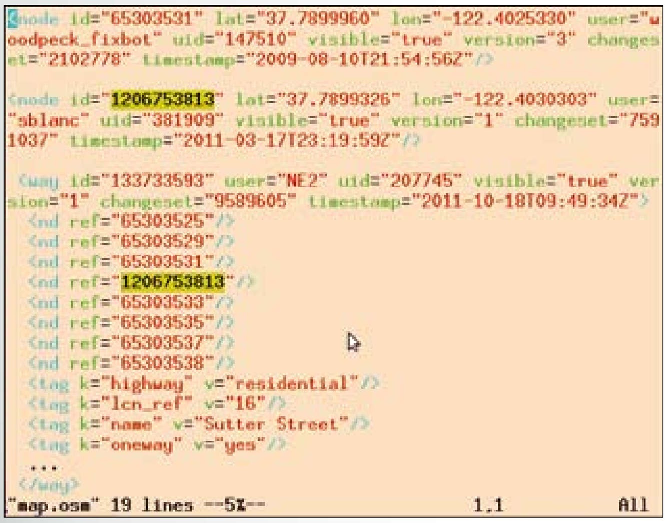


Figure 4: Freely licensed OSM data plugged into Google Maps.

area you are viewing. The `way` definition in the lower part of Figure 2 contains references to a total of eight nodes, denoted as `nd` elements in the XML. Two of them, `65303531` and `1206753813`, are visible as node definitions in the top part of the file.

Besides the geographical coordinates, `lat` and `lon`, or the geographical latitude and longitude in digital format, a node also lists who recorded it (`user`) and when (`timestamp`), as well as specifying the `changeset` in which the data were uploaded to the OSM server.

Just to compare the freely available data with what proprietary Google Maps has in store, you could take the latitude and longitude of the two nodes listed in the XML file in Figure 3 and plug them into Google Maps. You will see that it almost exactly matches the Sutter Street layout depicted in Figure 4.



```
<node id="65303531" lat="37.7899960" lon="-122.4025330" user="w
oodpeck_fixbot" uid="147510" visible="true" version="3" changes
et="2102778" timestamp="2009-08-10T21:54:56Z"/>

<node id="1206753813" lat="37.7899326" lon="-122.4030303" user=
"sblanc" uid="381909" visible="true" version="1" changeset="759
1037" timestamp="2011-03-17T23:19:59Z"/>

 <way id="133733593" user="NE2" uid="207745" visible="true" ver
sion="1" changeset="9589605" timestamp="2011-10-18T09:49:34Z">
  <nd ref="65303525"/>
  <nd ref="65303529"/>
  <nd ref="65303531"/>
  <nd ref="1206753813"/>
  <nd ref="65303533"/>
  <nd ref="65303535"/>
  <nd ref="65303537"/>
  <nd ref="65303538"/>
  <tag k="highway" v="residential"/>
  <tag k="lcn_ref" v="16"/>
  <tag k="name" v="Sutter Street"/>
  <tag k="oneway" v="yes"/>
  ...
 </way>
"map.osm" 19 lines --5%--                        1,1          All
```

Figure 2: The OpenStreetMap server exports map data as freely licensed XML.

Ways Make Streets

A street comprises one or multiple ways in the OSM database's XML. The `way` definition includes the name of the street to which it belongs in its `name` tag. The second to last XML `<tag>` line in Figure 3 thus assigns a value of `v="Sutter Street"` to the `k="name"` key of the way. Note that San Francisco's Sutter Street is longer than the way definition covered here. It is not uncommon that streets consist of several `way` definitions with identical name tags to allow for describing different street properties in different neighborhoods the street is winding through.

Flexible Strategy

The project has standardized the `way` tags shown in the XML in Figure 3: `highway` (what kind of street it is), `lanes` (number of lanes), `name` (the street name), and `oneway` (set to `"yes"` if it's a one-way street), but it also supports practically arbitrary extensions. The design is so flexible that new functions can be added quickly and unbureaucratically.

Although this hands-off approach encourages and empowers volunteer mappers, the disadvantages are uncontrolled extensions and significant data redundancy, which brings tears to the eyes of supporters of standardized database schemas. As the data gets interpreted worldwide by different applications, it needs to conform to agreed-upon standards, however. The undoubtedly required coordination process normally occurs on the OSM wiki and its Talk pages [2]. For example, if somebody suggests a new scheme for mailboxes or business opening hours, this typically prompts a heated discussion which often leads to a new standard.

Do-It-Yourself Mapping

Before you can start contributing, you need to register as a mapper by supplying your email address and creating a user account with a password. As on Wikipedia, mappers can make changes to the mapping data directly in the OSM

Mike: I couldn't find the parking. openstreetmap. org site. -rls

database, and data are immediately visible on the live server without any cross-checking. A number of editing tools are available – headed by the online editors JOSM [3] and Potlatch [4]. JOSM is fairly long in the tooth, and its cross-platform Java look is an interface that doesn't appeal to aesthetes, but it gets the job done. In contrast, Potlatch is a Flash application that runs directly in the browser; it has a polished finish and has become the standard for editing maps.

For this article, I will add parking and street-cleaning information for the streets of my neighborhood in San Francisco. Firing up any browser with Flash support and clicking the *Edit* button on openstreetmap.org, brings up an editable map. Selecting a street outlines the current `way` in yellow and shows its nodes in red (Figure 5). The left pane has *Simple* and *Advanced* tabs at the bottom. Selecting *Advanced* brings up all the way's tags, including the name of the street it describes.

Using the *Add* button, I added all the *parking* tags you see in Figure 5, and then I simply pressed the *Save* button at the top of the edit window, which asked me for a comment before submitting my changes to the server.

Wild West Parking Rules

Like any other major city in the world, San Francisco is completely mapped in OpenStreetMap. That said, San Francisco's complicated parking rules are a major challenge to automobile-driving tourists. It seemed worthwhile to integrate these bureaucratic regulations into the free world map.

If you have always thought that parking here is similar to the Wild West, you're wrong. On 23rd Street, around the corner from where I live, cars without a resident permit can park just two hours Monday through Friday from 8am to 6pm. Residents with a "Z" parking permit are the exception to this rule.

Additionally, the street sweeper [5] drives along most streets in San Francisco on certain days, and nobody is allowed to park on the side of the street that needs to be swept during that time. If you ignore the rule and park while the street sweeper comes by, you will be penalized with a US$ 55 parking ticket by one of the "Interceptor" cars following the sweeper. The street sign in Figure 6 shows you the details of the parking rules. The rules seem to differ from crossroad to crossroad, following an arcane strategy that only the city street sweeper department seems to know.

It might take awhile for San Francisco's parking rules to find their way into the OSM database for the whole city. However, it looks like the Germans are a step ahead here. You can see from Figure 7 that mappers have more or less recorded all the parking zones in the Bavarian town of Bamberg, as displayed by the *parking.openstreetmap.org* site, which is dedicated to parking information, showing parking discs at the appropriate locations. The wiki with the proposal for the street parking tag format [6] lists other cities that are advanced in this respect.

Parking Bureaucrats at Work

Parking wasn't standardized on OSM until recently, but after a lengthy discussion process, the tags shown in Figure 8 seem to have asserted themselves for street parking rules across the globe. The first three tags describe residential parking, the second block describes the street cleaning time for the left-hand side, and the third block does the same for the right-hand side of the street.

But how do you define "left" and "right"? Ways in OSM always point in one direction because they show you the way from one node to another. This is usually irrelevant for the mapped street (with one-way streets being the exception) and often simply reflects the arbitrary choice of the mapper.

In the case of San Francisco's parking rules, the direction allows you to define precisely when the street sweeper will

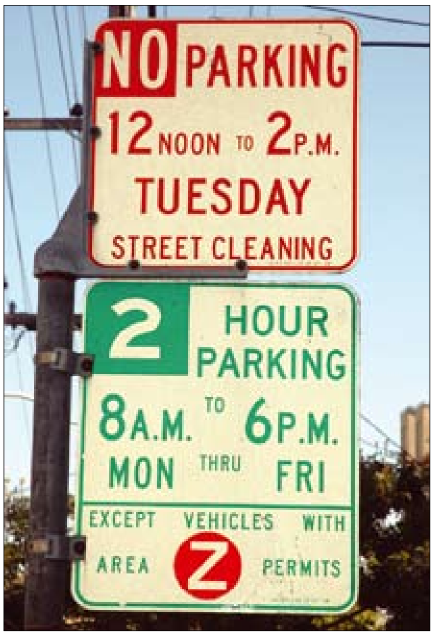
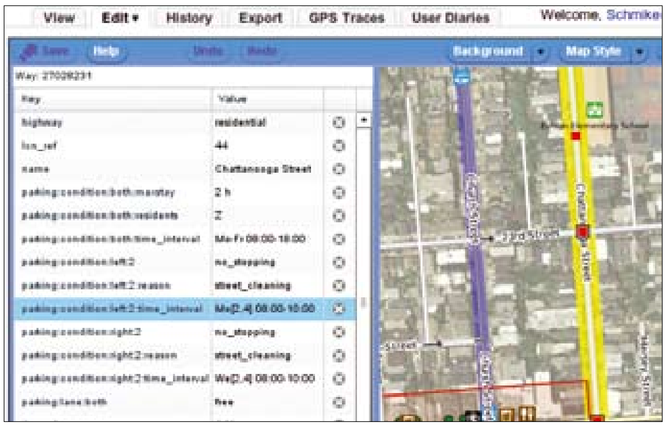


Figure 6: Resident parking with street sweeping every two weeks.



Way: 27028231

Key	Value
highway	residential
lcn_ref	44
name	Chattanooga Street
parking:condition:both:maxstay	2 h
parking:condition:both:residents	Z
parking:condition:both:time_interval	Mo-Fr 08:00-18:00
parking:condition:left:2	no_stopping
parking:condition:left:2:reason	street_cleaning
parking:condition:left:2:time_interval	Mo[2,4] 08:00-10:00
parking:condition:right:2	no_stopping
parking:condition:right:2:reason	street_cleaning
parking:condition:right:2:time_interval	We[2,4] 08:00-10:00
parking:lane:both	free

Figure 5: The Potlatch editor comes up after pressing the *Edit* button on openstreetmap.org.

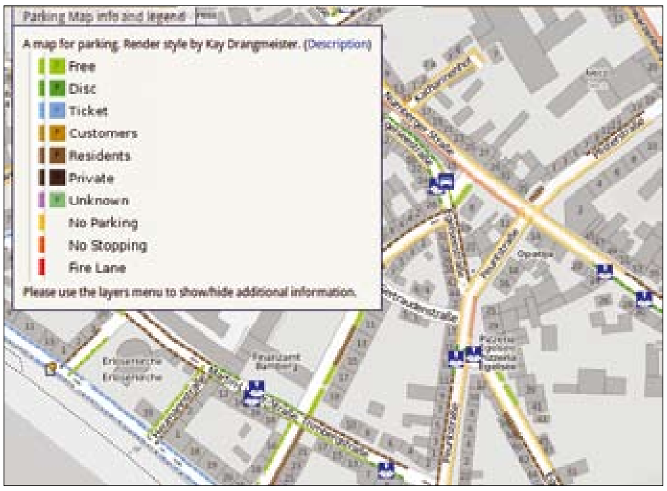
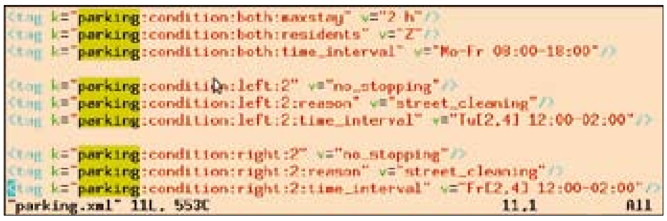

Parking Map info and legend
A map for parking. Render style by Kay Drangmeister. (Description)
Free
Disc
Ticket
Customers
Residents
Private
Unknown
No Parking
No Stopping
Fire Lane
Please use the layers menu to show/hide additional information.

Figure 7: The parking zones in downtown Bamberg are more or less completely recorded (parking.openstreetmap.org).



```
<tag k="parking:condition:both:maxstay" v="2 h"/>
<tag k="parking:condition:both:residents" v="Z"/>
<tag k="parking:condition:both:time_interval" v="Mo-Fr 08:00-18:00"/>

<tag k="parking:condition:left:2" v="no_stopping"/>
<tag k="parking:condition:left:2:reason" v="street_cleaning"/>
<tag k="parking:condition:left:2:time_interval" v="Tu[2,4] 12:00-02:00"/>

<tag k="parking:condition:right:2" v="no_stopping"/>
<tag k="parking:condition:right:2:reason" v="street_cleaning"/>
<tag k="parking:condition:right:2:time_interval" v="Fr[2,4] 12:00-02:00"/>
"parking.xml" 11L, 553C                                  11,1          All
```

Figure 8: After uploading the data to the OSM server, the whole world now knows when you can park on San Francisco's 23rd Street.

drive on which side of the street, in the case of different cleaning times for the two sides.

Figure 5 shows the Potlatch editor inserting a new tag by the name of `parking:condition:left:maxstay` into a way which marks sections of Chattanooga Street with a maximum parking time of two hours for cars without residential parking permit.

To make things even more complicated, in some neighborhoods, the street sweeper only comes around every second week of the month. Luckily, this can also be expressed using an OSM standard that has already been approved. Opening hours of businesses and government agencies also follow similar rules, which prompted OSM mappers to suggest using the existing syntax for parking rules. The wiki [7] explains who was responsible for this and when it happened. To make a long story short, the expression `"Fr[2,4] 09:00-11:00"` specifies the second and fourth Friday in the month for the street sweeper to come by between 9am and 11am.

Immediately Visible

After uploading, usually the whole world can see the change in the XML immediately and see it on the map after a few hours of delay. But the really exciting thing about this story is that any application can now use the newly recorded data freely. How about a script that tells me when the street cleaner is due if I tell it that I parked my car on the right-hand side of Chattanooga Street between 23rd and 24th Street?

The script in Listing 1 automatically downloads the XML data from the OSM server and outputs the right answer in just a couple of seconds:

LISTING 1: street-cleaning

```
#!/usr/local/bin/perl -w
#############################
# street-cleaning - Find
#   street cleaning times
#   between cross streets.
# Mike Schilli, 2012
# (m@perlmeister.com)
#############################
use strict;
use Geo::Parse::OSM;
use Graph::Directed;
use LWP::UserAgent;

my @bbox =
  qw( -122.4374  37.74754
      -122.42096 37.75894 );

my $url =
  "http://api.openstreetmap."
  . "org/api/0.6/map?bbox="
  . join ',', @bbox;

my $mapfile = "map.osm.gz";

my ($street_on,
    $street_cross1,
    $street_cross2, $side)
    = @ARGV;

die "usage: $0 street " .
    "cross1 cross2 side"
  if !defined $side;

my $ua =
  LWP::UserAgent->new();
$ua->default_header(
 "Accept-Encoding", "gzip");

if (!-f $mapfile
    or -M $mapfile > 7) {
 my $rsp = $ua->mirror($url,
                     $mapfile);
 $rsp->is_success
   or die $rsp->message();
}

my $osm = Geo::Parse::OSM->
  new( $mapfile );

my %on_nodes = ();

street_nodes(
 $osm, $street_on,
 sub {
   $on_nodes{ $_[0] } = 1; }
);

my $cross1_node =
  cross_find($osm,
 \%on_nodes, $street_cross1);
my $cross2_node =
  cross_find($osm,
 \%on_nodes, $street_cross2);

my ($nodes, $flip_order) =
  find_path_on_way($osm,
   $street_on, $cross1_node,
   $cross2_node);

$side = flipside($side,
                $flip_order);
my $parking = parking($osm,
    $nodes, $side);

print "Street Cleaning: ",
  street_cleaning($parking),
  "\n";

#############################
sub street_nodes {
#############################
 my ($osm, $name, $cb) = @_;

 $osm->seek_to(0);
 $osm->parse(
  sub {
   my ($n) = @_;
   if (exists
       $n->{tag}->{name} and
       $n->{tag}->{name} eq
       $name) {
    for my $n (
        @{ $n->{chain} }) {
     $cb->($n) or last;
    }
   }
  },
  only => "way"
 );
}

#############################
```

```
$ ./street-cleaning ↩
    "Chattanooga Street" ↩
    "23rd Street" ↩
    "24th Street" right ↩
    Street Cleaning: We[2,4] 08:00-10:00
```

For this to happen, Listing 1 first requests an XML file from the OSM server for the Noe Valley neighborhood in San Francisco. In a download form similar to the one shown in Figure 3, I've figured out that that my neighborhood covers an area between -122.4374 and -122.42096 degrees longitude and between 37.74754 and 37.75894 degrees latitude on the globe. Lines 18-21 in Listing 1 use this information to craft a URL for the OSM server API, and line 41 mirrors the compressed XML file as `map.osm.gz` on my local hard disk.

The *if* condition in lines 39 to 45 checks to see whether the file already exists and is no more than a week old, and it prevents the retransmission of rel-

LISTING 1: street-cleaning (continued)

```
103 sub cross_find {
104 #############################
105  my ($osm, $on_nodes,
106      $cross_street)    = @_;
107
108  my $found;
109
110  street_nodes( $osm,
111   $cross_street,
112   sub {
113    my ($n) = @_;
114    if ( exists
115         $on_nodes->{$n}) {
116     $found = $n;
117     return 0; # stop
118    }
119    return 1;  # continue
120   }
121  );
122
123  return $found;
124 }
125
126 #############################
127 sub find_path_on_way {
128 #############################
129  my ($osm, $way_name, @nodes)
130    = @_;
131
132  my $g =
133    Graph::Directed->new();
134
135  $osm->seek_to(0);
136  $osm->parse(
137   sub {
138    my ($n) = @_;
139    if (
140     exists $n->{tag}->{name}
141     and $n->{tag}->{name} eq
142     $way_name)
143    {
144     $g->add_path(
145           @{ $n->{chain} });
146    }
147   },
148   only => "way"
149  );
150
151  my $flip_order = 0;
152
153  my @path =
154    $g->SP_Dijkstra(@nodes);
155
156  if (!@path) {
157   @nodes = reverse @nodes;
158   @path =
159     $g->SP_Dijkstra(@nodes);
160   $flip_order = 1;
161  }
162
163  return (\@path,
164   $flip_order);
165 }
166
167 #############################
168 sub parking {
169 #############################
170  my ($osm, $nodes,
171      $side) =      @_;
172
173  my %to_match =
174    map { $_ => 1 } @$nodes;
175  my %results = ();
176
177  $osm->seek_to(0);
178  $osm->parse(
179   sub {
180    my ($w) = @_;
181
182    my @matches =
183      grep {
184        exists $to_match{$_}
185      } @{ $w->{chain} };
186
187    return if @matches < 2;
188
189    for my $tag (
190     keys %{ $w->{tag} }) {
191     if ($tag =~
192  /parking:condition:$side:.*/
193        ) {
194      $results{$tag} =
195        $w->{tag}->{$tag};
196     }
197    }
198   },
199   only => "way"
200  );
201
202  return \%results;
203 }
204
205 #############################
206 sub street_cleaning {
207 #############################
208  my ($parking) = @_;
209
210  for my $key (keys %$parking)
211  {
212   if ($key =~ /(.*)\:reason/)
213   {
214    if ($parking->{$key} eq
215     "street_cleaning") {
216     return $parking->{ $1
217        . ":time_interval" };
218    }
219   }
220  }
221
222  return undef;
223 }
224
225 #############################
226 sub flipside {
227 #############################
228  my ($side, $flip_order)= @_;
229
230  if ($flip_order) {
231   if ($side eq "left") {
232    $side = "right";
233   } else {
234    $side = "left";
235   }
236  }
237
238  return $side;
239 }
```

atively new data to speed up the process. The additional `Accept-Encoding` header in line 37 tells the server that the client prefers a gzipped file.

OSM Data Gobbler

The Geo::Parse::OSM module from CPAN picks its way through the XML data, and its `parse()` method expects a callback that it jumps to whenever it finds the element you are looking for. Note that `parse()` will not find anything if it has already run once and that `seek_to(0)` is required to return the parser to the start of the XML data file and thus launch a new search.

To answer questions about the parking conditions on a street between two cross streets, the script has to find one or more OSM ways located in this section of the street. In line 52, it uses the function `street_nodes` to find a street's node IDs and store them as keys in the `%on_street` hash. To accomplish this, the `street_nodes` function (defined in lines 80-100) searches through all of the ways in the XML file with the restriction `only => "way"` for a way that has the value of the given street in its `name` tag. As is typical for a parser, it uses a callback executed every time it finds something suitable.

Edsger Dijkstra to the Rescue

Armed with the nodes of the main street, the calls to `cross_find()` (defined as of line 103) in lines 59 and 62 find the nodes in which the crossroads defined on the command line intersect with the main street. This can get pretty complicated, however, if the distance between two intersections consists of several ways with many different nodes. Take nodes N2 and N5 in Figure 9, for example. Starting at N2, the algorithm now needs to move in one direction, hoping that it will arrive at N5, and not run up against a dead end in N1.

Fortunately, algorithm guru Edsger W. Dijkstra solved this problem back in 1959 [8], and the Graph module from CPAN sports the algorithm in its `SP_Dijkstra()` method. The latter only expects that two nodes are be connected somehow in a "Directed Acyclic Graph" (DAG) and then computes the shortest path from N2 to N5.

The `find_path_on_way()` function defined in lines 127 to 165 takes two arguments: the street name and an array of two nodes to connect it. It starts by creating a new Graph::Directed object and adds paths to it, connecting the nodes of every XML way definition covering the specified street.

To accomplish this, Geo::Parse::OSM finds a way's nodes by referencing the way's `chain` attribute. If the `SP_Dijkstra()` method can't find a path from A to B, the arbitrarily selected directional arrows between the nodes must be pointing in the wrong direction, and line 157 reverses the search direction, making a note of this fact in the `flip_order` variable, so that the main program later understands that "left" in this case doesn't mean "left" in the direction taken by the way, but "right."

Upon receiving the `find_path_on_way()` result, consisting of a valid node path between cross streets and a potentially modified `flip_order`, the `parking()` function browses through the path of nodes and stores the last parking tag discovered on the ways through which it travels. Assuming that the parking regulations don't change within a single street block, which is the case for street cleaning times, this provides valid information for querying users.

If a change of direction was initiated because no valid path could be found in the original directions, the `flipside()` function, defined as of line 226 and called in line 70, converts any `"left"` string into `"right"` and vice versa to help finding the correct side-specific parking rules. The `street_cleaning()` function, as of line 206 then simply needs to discover any existing parking tags, extract the appropriate side of the street ("left"/"right"), and output the data to tell tourists and residents alike wanting to park there when to expect the street cleaner.

Future

In the next couple of weeks, I'll carry on collecting the parking data in my home district, "Noe Valley" in San Francisco, and upload it to the server. I have a vision of a web application that lets me enter the current location of my street-parked second car, Perly Perlman, and notifies me by mail if the street cleaner with the parking ticket Piaggo [9] in its wake is due the next day. This is something that could really pay its way. ■■■

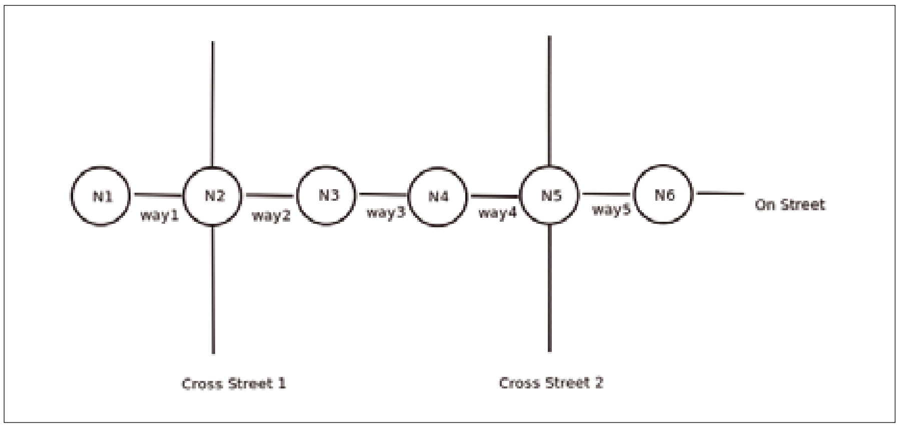


Figure 9: The Dijkstra Algorithm determines the shortest path between N2 and N5.

INFO

[1] Listings for this article: *http://www.linuxpromagazine.com/Resources/Article-Code*

[2] Discussion about the new parking tags on the OpenStreetMap wiki: *http://wiki.openstreetmap.org/wiki/Talk:Proposed_features/parking:lane*

[3] OSM editors, JOSM: *http://josm.openstreetmap.de*

[4] OSM editors, Potlatch: *http://wiki.openstreetmap.org/wiki/Potlatch_2*

[5] "Street Cleaning in San Francisco": *http://www.youtube.com/watch?v=kLR0uxooEf8*

[6] Wiki for the new parking tags: *http://wiki.openstreetmap.org/wiki/Proposed_features/parking:lane*

[7] Standards for business hours in OpenStreetMap: *http://wiki.openstreetmap.org/wiki/Key:opening_hours*

[8] "Dijkstra algorithm": *http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm*

[9] Parking ticket Piaggio in San Francisco: *http://usarundbrief.com/31/images/wespe.html*