Putting GPS data on maps

HIKE PILOT

Perl hackers take to the hills with a navigation system that provides a graphical rendering of a hiking tour. **BY MIKE SCHILLI**

Hiking with a navigation system is much more fun. A portable GPS device not only gives you your current position, but it can also tell you your altitude or the distance to a waypoint. Based on this data, the GPS device can also tell you your walking speed, the distance you have covered, and an estimated time of arrival. What's more, once you get back home you can attach your GPS device to your PC, download the data you collected en route, and map the hike.

Although the "eTrex" GPS receiver by Garmin is a low-budget device for beginners, it is perfect for the occasional hiker. The eTrex, which costs about 120 Euros (US$ 99 in the US), is handy, waterproof, and so robust that it will survive knocks and bumps without damage. To hitch up the eTrex GPS to your PC back home, you need a special interface cable that connects the eTrex to your computer's serial port. The official cable for the eTrex is quite expensive (at about 25 Euros, US$ 32, GBP 17), and this explains the project at [3], which helps you do it yourself. I must confess that I was lazy this time: I just bought it.

Babylonian Confusion

GPS receivers can use various formats for exported data, but a freeware program called *gpsbabel* [4] can help you fight the Babylonian confusion of tongues. Besides the ability to handle dozens of different data formats, the tool can read the data from a Garmin eTrex connected to the serial port of your Linux machine. The GPS receiver's memory stores the waypoints you've marked along the way, along with your routes (manually created collections of waypoints), and tracks (collections of coordinates recorded automatically every few seconds).

If you erase your track memory before you set off, and download it when you get back home, you will have an exact

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.

Figure 1: The "eTrex" mobile GPS receiver by Garmin is a low-budget device for beginners.

digital recording of your hike. This is a good starting point for various kinds of creative evaluations. Let's assume you connect your Garmin GPS to the second serial port on your PC; you could then enter **gpsbabel -t -i garmin -f /dev/ttyS1 -o gpx -F tracks.txt** to download the track data (-t) in Garmin format (-i garmin) from the second serial port (/dev/ttyS1, the first port would be /dev/ttyS0), and store the data in GPX format (-o gpx) in a file called tracks.txt (-F tracks.txt).

To avoid the need to run the process that controls the GPS device as root, you need to make the device entry for the serial port globally writable before reading the data (the process requires write privileges):

```
# chmod a+rw /dev/ttyS1
# ls -l /dev/ttyS1
crw-rw-rw- 1 root uucp 4, ➘
65 Feb 10 22:47➘
/dev/ttyS1
```

Some time later (be patient; serial ports were invented in the previous century), the *gpsbabel* command returns, and you will hopefully discover the XML-formatted track data in *tracks.txt* (Figure 3). To avoid the need to parse all the XML data for the evaluations we will be performing, the script in Listing 1 converts the data to YAML format, which is easier on the human eye than XML. Also, YAML data can incidentally be converted to a Perl data structure at one fell swoop. Figure 4 shows the YAML data – easier to read, don't you think?

track2yml uses the *XML::Twig* module from CPAN, which defines a handler that the Twig dancer jumps to for every *Trkpt* tag. The *XML::Twig::Elt* object passed to the handler represents the *<trkpt>* tag found with all of its sub-tags.

The *lat* (for latitude) attribute is a decimal value. Northern latitudes are positive and southern latitudes negative. The *lon* (for longitude) attribute expresses western longitudes as negative values, and eastern longitudes as positive values. The *ele* (for elevation) subelement gives you the height of the track point above sea level in meters. The *<time>* tag gives you the UTC time (GMT timezone) in ISO 8601 notation.

The *str2time()* function from the CPAN *Date::Parse* module converts the ISO 8601 timestamp to the timezone-independent Unix time in seconds for later calculations. The *handler()* function bundles all the data into a hash and

Figure 2: A special connector lets you hitch up the eTrex to your PC's serial port.

Listing 1: tracks2yml

```
#!/usr/bin/perl -w
use strict;
use Sysadm::Install qw(:all);

use XML::Twig;
use Date::Parse;
use YAML qw(DumpFile);

my $twig =
  XML::Twig->new(
 TwigHandlers =>
   { "trkpt" => \&handler, }
  );

my @points = ();
$twig->parsefile(
                "tracks.xml");
DumpFile("tracks.yml",
         \@points);

###################################
sub handler {
###################################
  my ($t, $trkpt) = @_;

  my $lat =
    $trkpt->att('lat');
  my $lon =
    $trkpt->att('lon');
  my $ele =
    $trkpt->first_child('ele')
    ->text();

  my $isotime =
    $trkpt->first_child(
   'time')->text();

  my $time =
    str2time($isotime);

  push @points,
    { lat     => $lat,
      lon     => $lon,
      ele     => $ele,
      time    => $time,
      isotime => $isotime,
    };
}
```

INFO

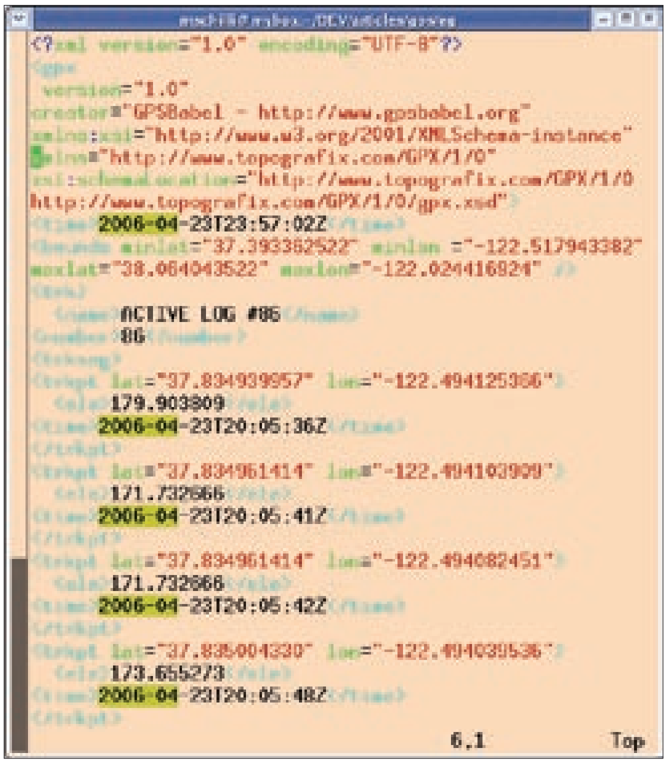
[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/69/Perl*

[2] Google Maps Hacks, Rich Gibson & Schuyler Erle, O'Reilly 2006

[3] HOWTO and mail address for do-it-yourself Garmin interface cable: *http://pfranc.com*

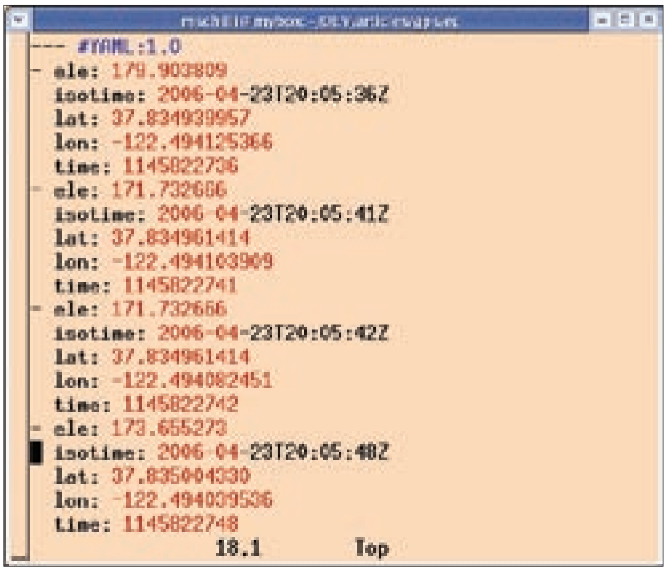
[4] GPS format converter: *http://www.gpsbabel.org*

[5] Yahoo! Maps Web Services – AJAX API Getting Started Guide: *http://developer.yahoo.com/maps/ajax/*



```
<?xml version="1.0" encoding="UTF-8"?>
<gpx
 version="1.0"
creator="GPSBabel - http://www.gpsbabel.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.topografix.com/GPX/1/0"
xsi:schemaLocation="http://www.topografix.com/GPX/1/0
http://www.topografix.com/GPX/1/0/gpx.xsd">
<time>2006-04-23T23:57:02Z</time>
<bounds minlat="37.393362522" minlon ="-122.517943382"
maxlat="38.064043522" maxlon="-122.024416924" />
<trk>
  <name>ACTIVE LOG #86</name>
<number>86</number>
<trkseg>
<trkpt lat="37.834939957" lon="-122.494125366">
  <ele>179.903809</ele>
<time>2006-04-23T20:05:36Z</time>
</trkpt>
<trkpt lat="37.834961414" lon="-122.494103909">
  <ele>171.732666</ele>
<time>2006-04-23T20:05:41Z</time>
</trkpt>
<trkpt lat="37.834961414" lon="-122.494082451">
  <ele>171.732666</ele>
<time>2006-04-23T20:05:42Z</time>
</trkpt>
<trkpt lat="37.835004330" lon="-122.494039536">
  <ele>173.655273</ele>
<time>2006-04-23T20:05:48Z</time>
</trkpt>
                                        6,1           Top
```

Figure 3: The GPX (XML) formatted track data I downloaded from the Garmin GPS.



```
--- #YAML:1.0
- ele: 179.903809
  isotime: 2006-04-23T20:05:36Z
  lat: 37.834939957
  lon: -122.494125366
  time: 1145822736
- ele: 171.732666
  isotime: 2006-04-23T20:05:41Z
  lat: 37.834961414
  lon: -122.494103909
  time: 1145822741
- ele: 171.732666
  isotime: 2006-04-23T20:05:42Z
  lat: 37.834961414
  lon: -122.494082451
  time: 1145822742
- ele: 173.655273
  isotime: 2006-04-23T20:05:48Z
  lat: 37.835004330
  lon: -122.494039536
  time: 1145822748
                    18,1         Top
```

Figure 4: The same track data in YAML format after converting with track2yml.

saves a hash reference as an element in the global array *@points*. The *YAML* module's *DumpFile* method, which is called later, stores the whole array, including the hash references, in an easily readable format in a file named *tracks.yml*, where subsequent scripts can read from it by calling *LoadFile()*.

Uphill and Down Dale

Today's hike takes us north of the Golden Gate Bridge along the "Coastal Trail" and the "Rodeo Trail" uphill and down dale through the Marin Headlands, a picturesque, hilly landscape on the Pacific coast. Taking the track data collected by the GPS system over the course of three hours, and plotting the elevation data against a time axis results in a graph like the one shown in Figure 5.

Shortly after 1:00 pm, I hit the trail at an altitude of about 200 meters above sea level, dropping down to sea level after about one and a half hours of gentle climbs and descents. This was followed by a steep 200 meter climb back to the starting point of the round course.

Listing 2 plots the graph using the *RRDTool::OO* module, which uses the *rrdtool* round-robin database under the hood. I used *rrdtool* due to its elegant (say: automatic) date display on the X axis. Line 8 reads the YAML data, and the following *new()* constructor creates a new RRD database, using a temporary file, as we will not need the data later. The *tmpfile()* function returns two arguments, of which we will only be passing the first one to *new()*.

The *create()* method then defines the data store schema, which expects a value every 60 seconds. The GPS receiver delivers track data every couple of seconds, but *rrdtool* simply aggregates

Listing 2: elerrd

```
01 #!/usr/bin/perl -w
02 use strict;
03 use YAML qw(LoadFile);
04 use RRDTool::OO;
05 use File::Temp qw(tempfile);
06
07 my $trkpts =
08   LoadFile("tracks.yml");
09
10 my $rrd =
11   RRDTool::OO->new(
12   file => (tempfile())[1]);
13
14 $rrd->create(
15   start =>
16     $trkpts->[0]->{time} - 1,
17   step        => 60,
18   data_source => {
19    name => "elevation",
20    type => "GAUGE"
21   },
22   archive => { rows => 10000 }
23 );
24
25 for my $trkpt (@$trkpts) {
26   eval {     # Deal with dupes
27    $rrd->update(
28     time  => $trkpt->{time},
29     value => $trkpt->{ele}
30    );
31   };
32 }
33
34 $rrd->graph(
35   start =>
36     $trkpts->[0]->{time},
37   end =>
38     $trkpts->[-1]->{time},
39   image => "elevation.png",
40   vertical_label =>
41     'Elevation',
42   width       => 300,
43   height      => 75,
44   lower_limit => 0,
45 );
```

the data. The database can store a maximum of 10,000 averaged one-minute elevation values, and that should be enough for even extended hikes.

The *for* loop starting in line 25 iterates over the trackpoints and feeds them into the database, along with the track time, using the *update()* method. As *rrdtool* complains and aborts if the same timestamp occurs twice, I wrapped the update command in an *eval* block to teach the script to be more tolerant.

The *graph()* method plots the graph. The first trackpoint sets the start time, and the timestamp for the last trackpoint sets the end time. In next to no time, you should have an attractive PNG-formatted diagram in the file specified in Line 39, *elevation.png* (Figure 5).

Sums and Spheres

To calculate the distance covered, Listing 3 has to iterate through all the trackpoints, calculate the distance between them, and add the individual distances. Each trackpoint is a reference to a hash that stores the latitude in *lat* and the longitude in *lon*. *$last_pt* stores the trackpoint from the last round of the loop (apart from in the first round). It calculates the distance between two trackpoints based on known longitudinal and latitudinal values. This is nontrivial, as the values represent points on the surface of an ellipsoid. The CPAN *Geo::Distance* module uses trig functions to perform the calculations and provides a simple *distance()* method, which expects the required unit of distance ("kilometer" or "mile") and two trackpoints as longitudinal and latitudinal values. The function returns a value for the distance, which *dist* then gobbles up:

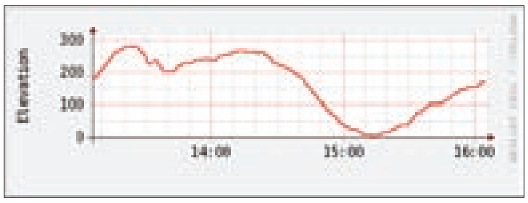


Figure 5: Elevation above sea level during the hike.

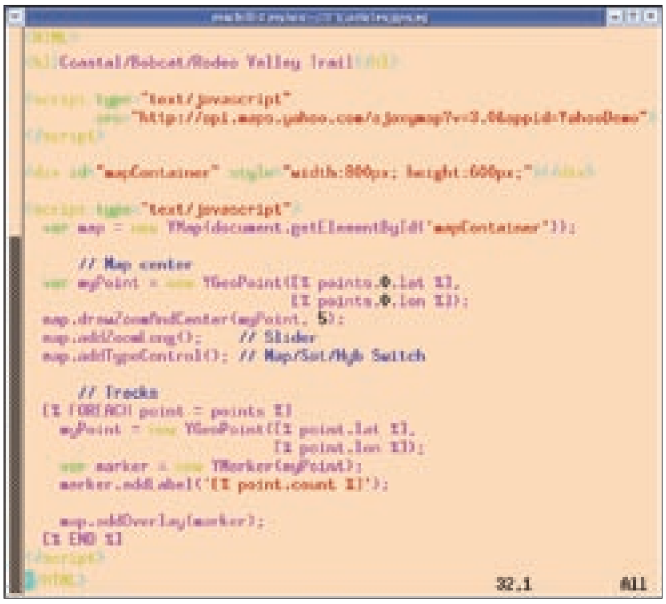
```
$ ./dist
Total: 11.67km
```

Mashing Up Maps

Recently so called "mash-ups" have been all the rage; basically, this means pimping an online map with a do-it-yourself extension. Besides Google, the company I work for, Yahoo!, also lets programmers dynamically add tags to scalable maps using a simple Javascript API.

To plot the trail on a Yahoo map, I first have to reduce the volume of the data. The 1800 trackpoints I collected during the hike would just give me an unintelligible mess. This is why Listing 4 iterates over the trackpoints in a *for* loop, pushing trackpoints that are more than 0.4 kilometres from the preceding trackpoint to the back of the *@points* array. Again, *Geo::Distance* takes care of the complex distance calculations.

A Mash-Up HOWTO is avaiable from [5]. Figure 6 shows you the required Javascript code. If you intend to use the API, note that you should obtain an application ID first. Listing 4 uses the *YahooDemo* ID, which allows 50,000 requests per day for an IP address. Note that the service doesn't permit live GPS navigation; the GPS data has to be at least 6 hours old.



```
<html>
<h1>Coastal/Bobcat/Rodeo Valley Trail</h1>

<script type="text/javascript"
        src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=YahooDemo">
</script>

<div id="mapContainer" style="width:800px; height:600px;"></div>

<script type="text/javascript">
 var map = new YMap(document.getElementById('mapContainer'));

    // Map center
 var myPoint = new YGeoPoint([% points.0.lat %],
                             [% points.0.lon %]);
 map.drawZoomAndCenter(myPoint, 5);
 map.addZoomLong();     // Slider
 map.addTypeControl(); // Map/Sat/Hyb Switch

    // Tracks
[% FOREACH point = points %]
  myPoint = new YGeoPoint([% point.lat %],
                          [% point.lon %]);
  var marker = new YMarker(myPoint);
  marker.addLabel('[% point.count %]');

  map.addOverlay(marker);
[% END %]
</script>
</html>
                                                     32,1          All
```

Figure 6: The template with the Javascript code for creating the mash-up.

The Javascript code and various HTML tags are stored in the *map.tmpl* template file, which is read by Listing 4, interpreted via the CPAN Template Toolkit by processing the content within the magical *[%...%]* tags. The Template Toolkit provides a simple scripting language with only limited control functionality to avoid having too much program logic in the presentation layer. Also, access to variables is amazingly simple; hashes, arrays, and references are all handled in the same way using a magic

Listing 3: dist

```
#!/usr/bin/perl -w
use strict;
use YAML qw(LoadFile);
use Geo::Distance;

my $trkpts =
  LoadFile("tracks.yml");
my $geo =
  Geo::Distance->new();

my $total = 0;
my $last_pt;

for my $trkpt (@$trkpts) {
  if ($last_pt) {
    my $k = $geo->distance(
      "kilometer",
      $last_pt->{lon},
      $last_pt->{lat},
      $trkpt->{lon},
      $trkpt->{lat}
    );

    $total += $k;
  }
  $last_pt = $trkpt;
}

printf "Total: %.2fkm\n",
       $total;
```

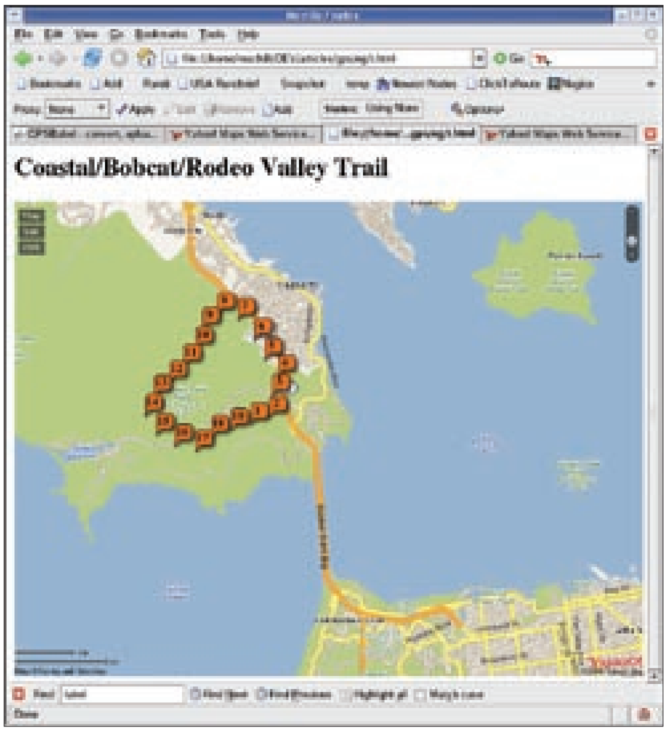


Figure 7: The finished mash-up with trackpoints from the hike north of San Francisco. A script plots points recorded on the hike by the eTrex device

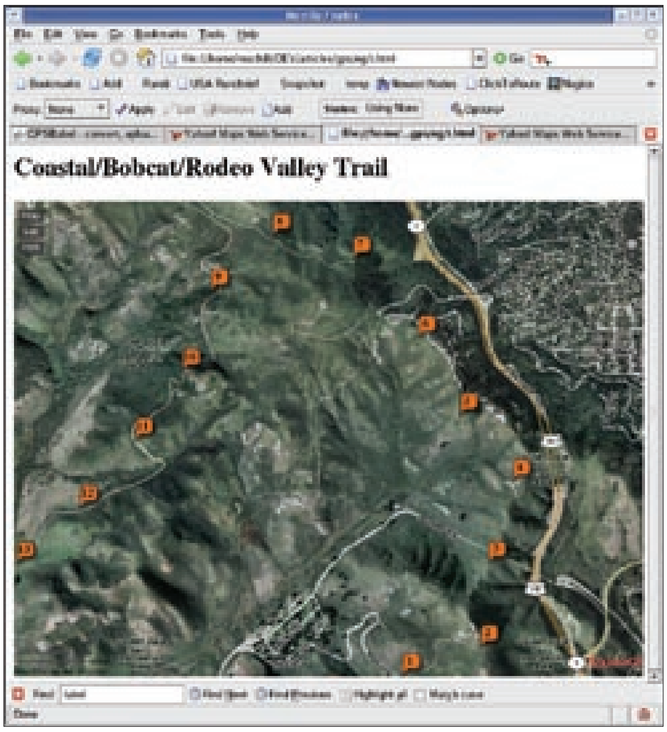


Figure 8: The same mash-up after clicking the button on the top left to toggle to the hybrid satellite image, then scaling up using the zoom tool.

dot (.). For example, to reference the first element in an array pointed to by *$points*, and to extract the value for the *lat* key in the underlying hash, the Template Toolkit notation is *points.0.lat*. Perl would need *$points->[0]->{lat}* for this. Sweet!

The output from Listing 4 is simply redirected to an HTML file and then rendered by a browser. This gives you a 600x400 pixel window with a map that you can pan and scale, since the embedded Javascript talks to Yahoo's map server. You can toggle to the satellite view, and there is even a *hybrid* mode that allows you to overlay the satellite image with data from the map.

Figure 7 shows the initial browser image in which the trackpoints are displayed as little orange bubbles numbered 1 through 19. Readers outside the US should note that Yahoo maps for your country may not be as detailed, but (low-res) satellite images should be available at least.

The Javascript code in Figure 6 only demonstrates some of the Map API's simplest gimmicks; you can add bubbles with images and all kinds of other goodies. Events such as mouse clicks and drags can be captured, evaluated using Javascript code, and possibly sent back to the server using Ajax tricks. ■

Listing 4: map

```
#!/usr/bin/perl -w
#################################
# map - Put track markers on
#       a Yahoo Map
#################################
use strict;
use YAML qw(LoadFile);
use Geo::Distance;
use Template;

my $trkpts =
  LoadFile("tracks.yml");
my $geo =
  Geo::Distance->new();

my $count = 0;
  # Minimum marker distance
my $min   = 0.4;
my @points = ();
my $last_pt;

for my $trkpt (@$trkpts) {
 if ($last_pt) {
  my $k = $geo->distance(
    "kilometer",
    $last_pt->{lon},
    $last_pt->{lat},
    $trkpt->{lon},
    $trkpt->{lat}
  );

  next if $k < $min;
 }
 $trkpt->{count} = ++$count;
 push @points, $trkpt;

 $last_pt = $trkpt;
}

my $template =
  Template->new();
my $vars =
  { points => \@points };

$template->process(
  "map.tmpl", $vars)
  or die $template->error();
```