

MAPPING HACKS™

100 Industrial-Strength Tips & Tools



O'REILLY®

*Schuyler Erle,
Rich Gibson & Jo Walsh*

HACK
#58

Don't Lose Your Tracklogs!

Tracklogs are the stuff of your stories. Here are simple ways to preserve them for future use, and a harder but more satisfying way to annotate your tracks.

GPS tracklog memories are great, until you run out of space. On one trip, I had to decide whether I'd prefer to have some data for the whole trip, but lose date and timestamps, or just lose the data for the end of the trip. That choice continues to haunt me. Once you get started collecting tracklogs and you explore their possibilities, you may become fanatically obsessed about preserving these stories from your travels!

There are two risks associated with saving your tracklogs. The obvious risk is running out of space. A typical GPS unit may store approximately 1,500 track points, or 25 minutes of points if you store points at the admittedly extreme rate of one point per second. Higher-end units, like the Garmin Legend and Vista hold 10,000 points, or almost three hours at one point a second. It would be more common to store points less frequently, so those 1,500 points are quite useful, and 10,000 is almost an acceptable number for those tracklog freaks. (Some recent GPS devices are offering 50,000 and more, though 10,000 is still common.)

Sometimes a shortage of track memory leads to odd predicaments. On several trips, I've brought an old laptop solely to download tracklogs because I didn't have a USB cable to connect my serial-only GPS to my USB-only iBook. I also didn't have a way to get my tracklogs off my old laptop and onto the iBook. I didn't have a way, that is, until I remembered that I had a PCMCIA-to-Compact Flash adapter that worked great on the old laptop. I copied the tracklogs to the CF card and then put the card into my digital camera, which I connected to the iBook with a USB cable. The camera happily reported itself to the iBook as a mass storage device, allowing me to copy off the tracklogs.

The more insidious risk of tracklog preservation is thinking that you have space that you actually don't. Garmin GPS units have an active tracklog, plus a number of "saved" tracklogs. It would be reasonable to think you could save your active log in one of the saved logs. Sadly, this doesn't work. When you save a tracklog, the software compresses the points, getting rid of duplicates and discarding the date and timestamps. This makes sense if you are [using the saved tracks for your own base maps \[Hack #55\]](#), or as the data for the Track Back feature, where the GPS will guide you to retrace your steps to take you home. But if you care about the temporality of your track points, you don't want to save your tracklogs in your GPS.

GPS Data Loggers

What if 10,000 track points aren't enough for you? You need a Tracklog Logger. These devices connect to your GPS and continually log the GPS output. They understand the National Marine Electronics Association (NMEA) standard and will thus work with almost any GPS that has a serial port.

Prairie Geomatics (<http://www.prairie.mb.ca/>) sells a line of GPS data loggers. The “basic” DGPS-XM4 can store position, date, and time for 393,000 points. At one sample per second, that is about four and a half days of tracklogs. The SRVY-XM4-ALT costs the same amount, also tracks altitude, and stores 308,800 points. Prairie Geomatics also offers the DGPS-XM4-ME, shown in Figure 5-31.



Figure 5-31. Prairie Geomatics DGPS-XM4-ME

This device will store position, date, and time, plus it has five “event” buttons on the front. Each time a button is pushed, it stores a timestamped position with a label identifying which button you pushed. This allows you to quickly mark positions. For example, you could do population surveys while in a moving vehicle, or you could use it to mark noteworthy items while traveling.

Hacking the Hack: Build Your Own Data Logger

Being a geek, you probably read that section and thought “Ah, I could make one of those.” Well of course you can! You can get the same effect by connecting your GPS unit to a laptop and logging everything that comes through. If you have a budget and an assigned project that requires a solid “solution,” then the GPS data loggers are smaller, consume less power, and are less complicated. But it is more fun to make your own!

You could write a small program to directly open the serial port and log the GPS output, but that means that only one program at a time can get position information. The answer? Run *gpsd* [Hack #57] and then run a program to catch and log the position information that it reports.

Here is a program called *gpslogger.pl* that connects to the *gpsd* process and logs position, and a little bit more:

```
#!/usr/bin/perl

use 5.6.1;
use IO::Socket;
use Term::ReadKey;
use strict;

BEGIN { $|++ }; # autoflush STDOUT.

# Set parameters based on command line options, with defaults.
my $Rate      = 1;
my $extended  = shift( @ARGV ) || 0;
warn "extended: $extended\n";
my $GPSD      = shift( @ARGV ) || "localhost:2947";
# Connect to gpsd.
my $gps = IO::Socket::INET->new($GPSD)
    or die "Can't connect to gpsd at $GPSD: $!\n";

while (1) {
    my $key;
    if ($extended) {
        warn "Enter an annotation, followed by <enter> \n";
        $key = <>;
        chomp $key;
    } else {
        ReadMode 'cbreak';
        $key = ReadKey(-1);
    }
    # Tell gpsd we want position, altitude, date, and status.
    $gps->print("pads\n");

    # Parse out the response. If the date is blank, gpsd needs
    # a second to catch up.
    my $location = <$gps>;
```

```

my ($lat, $long, $alt, $date, $status) =
    ($location =~ /P=(.+?) (.+?),A=(.+?),D=(.+?),S=(.+?)\/gos);

next unless $status or $key;

# Print a line of data to both STDOUT and STDERR, and wait.
print $_ join("|", $lat, $long, $alt, $date, $key || "" ), "\n"
    for ( \*STDOUT, \*STDERR );

sleep( $Rate );
}

```

Start the program:

```
$ ./gpslogger.pl > annotation.txt
```

```

38.414367|-122.790083|87.100000|2004-10-23T23:25:33Z|A
38.414467|-122.789900|87.100000|2004-10-23T23:25:34Z|
38.414617|-122.789650|87.100000|2004-10-23T23:25:35Z|D

```

The output is pretty obvious: lat, long, elevation, and a timestamp, all delimited with the pipe symbol (`|`). But what is that character at the end of the line? That is the homebrew answer to the DGPS-XM4-ME's "event" buttons. When `log_position.pl` is running in the foreground and you press a key, it logs that key, the date and time, and your current location. Later, you can map the points with color codes for the different keys.

We can use the *shapelib* utilities [Hack #92] to create a script to convert this output to a shapefile that contains the tracklog, timestamp, and the buttons that you pushed:

```

#!/bin/sh
shpcreate events point
dbfcreate events -n elevation 5 0, -s timestamp 19, -s event 5
grep -v "^#" $1 | \
awk -F'|' '{print "shpadd events " $1 " " $2 "; dbfadd events \"\" $3 \"\" \"\" $4 \"\" \"\" $5 \"\""}' | \
sh

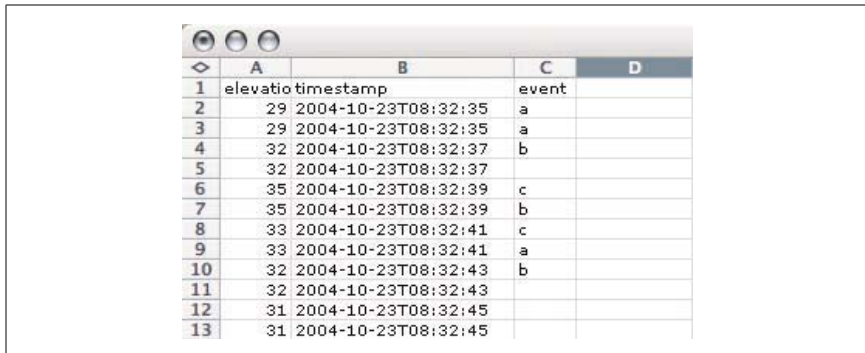
```

To run the script, use this command:

```
$ ./evt2shp.sh data.txt
```

This will create a shapefile named *events*. This really refers to three files: *events.shp*, *events.shx*, and *events.dbf*. The file *events.dbf* is a dBase-format file that can be read in many different programs. The elevation, timestamp, and event fields are shown in Excel in [Figure 5-32](#).

To see the contents of the *dbf* file from the command line, use the *dbfdump* utility from the *shapelib* utilities: **dbfdump events.dbf**. The latitude and longitude are stored in the *shp* and *shx* files and can be viewed by running the *shpdump* utility from *shapelib*: **shpdump events.shp**.



	A	B	C	D
1	elevation	timestamp	event	
2	29	2004-10-23T08:32:35	a	
3	29	2004-10-23T08:32:35	a	
4	32	2004-10-23T08:32:37	b	
5	32	2004-10-23T08:32:37		
6	35	2004-10-23T08:32:39	c	
7	35	2004-10-23T08:32:39	b	
8	33	2004-10-23T08:32:41	c	
9	33	2004-10-23T08:32:41	a	
10	32	2004-10-23T08:32:43	b	
11	32	2004-10-23T08:32:43		
12	31	2004-10-23T08:32:45		
13	31	2004-10-23T08:32:45		

Figure 5-32. The *dbf* file from a *shapefile*

Our program logs “key presses,” but if we salvage the keyboard encoder from a surplus keyboard, we can push the meaning of a “keyboard” well past the breaking point. There is a whole culture of people creating arcade game emulators using the Multiple Arcade Machine Emulator (MAME) software. These folks have created a full literature on using the electronics from old keyboards in new and entertaining ways as general data-input devices. Do a Google search on “hacking keyboard encoders” for more information.

We can put this all together to create a compact and efficient portable tracklog-logging and event-logging system. In Figure 5-33, we see one such system. The computer is a Fujitsu Stylistic obtained at a surplus price. This is a 486-based tablet machine running the Pebble distribution of Linux. (Note: Pebble defaults to running *getty* on the serial port, so in order to get *gpsd* to work, you must comment out all lines in your */etc/inittab* that refer to your serial port: */dev/ttyS0* on the Stylistic.) A Garmin GPS is connected via a serial cable. The Stylistic has an external keyboard connector, so we are using the “little man” keyboard. Pressing the buttons that make up his face generates keystrokes. (We could also use the Happy Hacker keyboard on the steering wheel if we wanted to be a tad less photogenic.)

Using this setup, we can input GPS data and then map our results in terms of button pushes of the “left eye,” “right eye,” “nose,” and “mouth parts 1 to 4.” These work the same as the Track log Logger event buttons.

The *gpslogger.pl* program has one other trick: it can do extended annotations. If you start it with a command-line parameter, it will pause and wait for you to create an annotation that ends with the <enter> key:

```
./gpslogger.pl yes > extended.txt
```

```
Enter an annotation, followed by <enter>
```

```
Fireman with dog
```

```
38.403367|-122.829733|55.100000|2004-10-26T01:13:24Z|Fireman with dog
```



Figure 5-33. A homebrewed geo-annotation device with custom “little man” keyboard

The extended annotations can be converted to shapefiles with (almost) the same script as the previous one. The one change is to increase the size of the event field. In the line:

```
dbfcreate events -n elevation 5 0, -s timestamp 19, -s event 5
```

change `-s event 5` to `-s event 1024`. This number (1024) needs to be larger than your longest annotation. In Figure 5-34, we see part of the results of extended geo-annotation. Note how the dots that mark annotations are slightly offset from the street.

This package, along with the saw blade at the base of the “keyboard,” makes a perfect data-logging and geo-annotating solution, except for the possible difficulty in getting through airport security and the always possible risk of personal injury. What if we could use an ultra small computer? The Gumstix Waysmall computers (<http://gumstix.com/>) are tiny!

They are full Linux computers running on the Intel XScale PXA255 processor. They have two serial ports, one that you normally reserve to connect to the console in order to control the unit, and the second free and waiting to

