## Getting into the Banshee metadata

# Hot Beats

**The Banshee music player stores song metadata in an SQLite database that a Perl script can query and manipulate. We'll whip up a quick backup and restore script and look into a new algorithm calculating beats per minute.** *By Michael Schilli*

Just like the choice between vi or Emacs, many users swear by their favorite music player and are reluctant to change. After all, it isn't easy to transfer all those playlists and ratings that you painstakingly have put together over the years. Although I will never use any editor except vi, I did recently try out the Banshee music player [1], because Rhythmbox, which I had previously used, did not offer a simple way to export ratings.

The GUI looks extremely clean and well thought out (Figure 1), and when I discovered that you can easily save, export, or externally manipulate the ratings you enter in Banshee – because the player stores them in an easily accessible SQLite database – I fell head over heels. Thus, operation "Player Change" was launched. As you can see in Figure 2, Banshee stores song metadata in the `CoreTracks` table of the `~/.config/banshee-1/banshee.db` database file.

Mike: Only 2 slashes instead of 3 correct?? -rls

The path to the referenced audio files on the filesystem sits in the `Uri` column with a URI prefix of `file://`. As the SQLite `.schema` command shows, the table has many more interesting columns, such as the number of rating stars (`Rating`) or the number of bass drum hits per minute that get disco dancers moving ecstatically on the dance floor – otherwise known as `BPM`, or beats per minute.

**Figure 1:** The Banshee music player.

## Saving Your Ratings

Listing 1 backs up all the song ratings you have meticulously clicked and reads out the database, compiling a map of songs to rating values in a YAML-formatted file (Figure 3), which you can restore later in case of a system crash or if you change to another music player.

If you call the `banshee-rating-backup` script without any parameters, it opens the database with the DBI module and – in the `backup` function starting in line 45 – uses a `SELECT` command to iterate over all the entries in the `CoreTracks` table. If the rating for a song is equal to `0`, line 59 jumps to the next entry, because there is no need to save non-existent ratings.

However, if the script finds a positive value, line 63 saves it in the `%ratings` hash under a key that's the path of the audio file. After completing all table entries, the `DumpFile()` function, courtesy of the YAML module, writes the ratings to the YAML file `banshee-ratings.yml`.

To restore the ratings at a later time, you just need to call the `banshee-rating-backup` with the `-r` option, which calls the `restore` function starting in line 74. It uses YAML's `LoadFile()` function in line 79 to read the `banshee-ratings.yml` file and then iterates over the entries of the hash initialized by this action.

The keys in the hash are the path names to the rated audio files, and its values are the ratings. Line 89 simply needs to issue an SQL update command to restore the database to its full glory, one record at a time. The typical hop, step, and jump with the DBI module involves putting together the SQL query with `prepare()`, followed by an `execute()` with parameters, which replaces the variables represented by question marks in the query. Finally, a `finish()` releases the allocated statement handle `$sth`.

## Beats per Minute

Ratings alone are not enough to create a playlist for specific events or moods – would you really want to listen to AC/DC at an intimate dinner? Banshee has a "BPM" field in 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*.

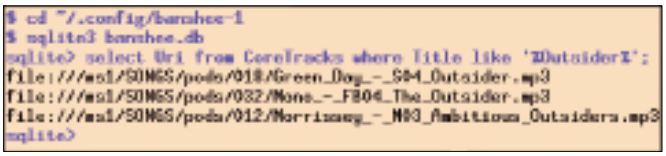



**Figure 2:** The open database design makes Banshee an open book.


**Figure 3:** Excerpt of YAML file with Banshee user ratings.

metadata database to provide more orientation.

The booming disco bass of a party track like "Memories" by David Guetta will clock about 120 BPM, and a fast techno track about 180. However, a classical piece like "The Magic Flute" by Mozart entirely does without drums and thus scores low BPM values. Depending on how you interpret the BPM definition, this might not be entirely correct because even classical pieces define a regular beat; however, for my simplified candlelight dinner compatibility meter, I'll assume a BPM value of 0 for drumless pieces. Radio DJs swear by this value and, in some cases, use automated tools to put together a BPM-compatible program.

With a combination of the rating and the permitted BPM range, users can choose the appropriate musical underpinnings for those special moments. Unfortunately, audio files don't normally include BPM values in their metadata.

Version 1.5 of the Banshee player includes a BPM detection tool based on the GStreamer bpmdetect package. Checking a box in *Preferences* | *Source Specific* enables the BPM detector (Figure 4) and populates the database with BPM values. The *Tools* | *Rescan Music Library* item starts the CPU-hungry update – best to be run overnight.

Unfortunately, the results leave much to be desired: The fairly laid back "I Feel Fine" by The Beatles scores a heady 213


**Figure 4:** Banshee can calculate BPM values.

## LISTING 1: banshee-rating-backup

```
01 #!/usr/local/bin/perl -w
02 ##############################
03 # banshee-ratings-backup
04 # Mike Schilli, 2011
05 # (m@perlmeister.com)
06 ##############################
07 use strict;
08 use Log::Log4perl qw(:easy);
09 Log::Log4perl->easy_init(
10  $DEBUG);
11
12 use DBI qw(:sql_types);
13 use DBD::SQLite;
14 use Data::Dumper;
15 use YAML
16   qw(LoadFile DumpFile);
17 use Getopt::Std;
18
19 getopts("r", \my %opts);
20
21 my $db = glob "~/.config" .
22      "/banshee-1/banshee.db";
23 my $dbh = DBI->connect(
24  "dbi:SQLite:$db",
25  "", "",
26  {
27   RaiseError => 1,
28   AutoCommit => 1
29  }
30 );
31
32 my $yml =
33   "banshee-ratings.yml";
34 my %ratings = ();
35
36 if ($opts{r}) {
37  restore($yml, $dbh);
38 } else {
39  backup($dbh, $yml);
40 }
41
42 $dbh->disconnect();
43
44 ##############################
45 sub backup {
46 ##############################
47  my ($dbh, $yml) = @_;
48
49  my %ratings = ();
50
51  my $sth = $dbh->prepare(
52   "SELECT * FROM CoreTracks"
53  );
54  $sth->execute();
55
56  while (my $hash_ref =
57   $sth->fetchrow_hashref())
58  {
59   next
60     if $hash_ref->{Rating} ==
61      0;
62
63   $ratings{ $hash_ref
64      ->{Uri} } =
65     $hash_ref->{Rating};
66  }
67
68  DumpFile($yml, \%ratings);
69
70  $sth->finish();
71 }
72
73 ##############################
74 sub restore {
75 ##############################
76  my ($yml, $dbh) = @_;
77
78  my $ratings =
79    LoadFile($yml);
80
81  for
82    my $song (keys %$ratings)
83  {
84   DEBUG "Restoring $song";
85
86   my $rating =
87     $ratings->{$song};
88
89   my $sth = $dbh->prepare(
90     "UPDATE CoreTracks " .
91     "SET Rating = ? " .
92     "WHERE Uri = ?");
93   $sth->execute($rating,
94    $song);
95   $sth->finish();
96  }
97 }
```

BPM points – three times more than the superfast pop/punk track "Rich Lips" by Blink-182, which only scores 68 BPM (Figure 5).

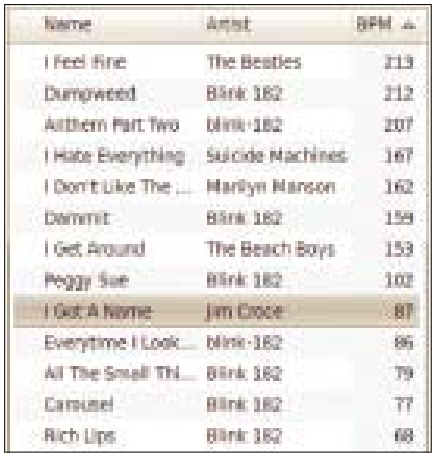
In Listing 2, I am thus trying to find a more reliable BPM counter solution. To do so, the program uses the `sox` utility from the sox package in Ubuntu to convert the compressed audio files into raw audio data, runs them through a narrow bandpass filter in the bass range, and then measures what will hopefully be the bass peaks. Although this method isn't foolproof, it can at least distinguish hoof-stamping disco tracks from classical music.

The Audacity tool, which is available as an audio tool package for many distributions, shows what the audio data for two different musical genres looks like (Figure 6). Although the classical orchestral piece with a heroic tenor shows only slight deflection, you can easily see the periodic tendencies in the broadband synthesizer sound.

## Sampled Music

Music, which people's ears perceive as a bundle of sound frequencies played at the same time and in harmonic pitches, are created from digital sampling values by the sound card. A stereo recording typically comprises 44,100 different 16-bit samples per second, with values that range between -32,768 and +32,767. A pure tone would look very much like a sine wave, but a musical instrument or a human voice typically generates a wide spectrum of frequencies.

An MP3 file applies a sophisticated encoding method to the sampled values, and the script first needs to convert them to raw format before analysis can be performed. For this to happen, I need the `sox` utility:



| Name | Artist | BPM |
|---|---|---|
| I Feel Fine | The Beatles | 213 |
| Dumpweed | Blink 182 | 212 |
| Anthem Part Two | blink-182 | 207 |
| I Hate Everything | Suicide Machines | 167 |
| I Don't Like The ... | Marilyn Manson | 162 |
| Dammit | Blink 182 | 159 |
| I Get Around | The Beach Boys | 153 |
| Peggy Sue | Blink 182 | 102 |
| I Got A Name | Jim Croce | 87 |
| Everytime I Look ... | blink-182 | 86 |
| All The Small Thi ... | Blink 182 | 79 |
| Carousel | Blink 182 | 77 |
| Rich Lips | Blink 182 | 68 |

**Figure 5:** Banshee's BPMs are left wanting.

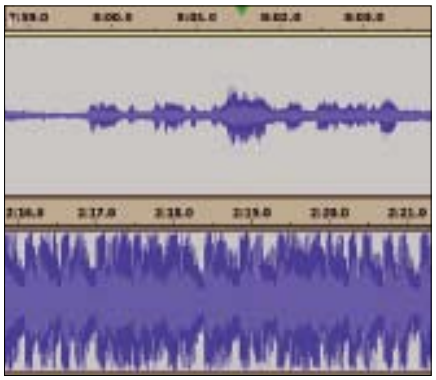


**Figure 6:** "Zu Hilfe, zu Hilfe, sonst bi-i-in ich verloren" (Help, help, otherwise I am lost) trills Prince Tamino in Mozart's "The Magic Flute" (top) compared with the full synthesizer sound of "Memories" by David Guetta (bottom).

```
sox infile.mp3 -r 44100 -c 2 \
-b 16 -t raw -e signed outfile.raw
```

This code creates the `outfile.raw` file in raw format as a two-channel encoding with signed 16-bit values at a sampling rate of 44,100Hz from `infile.mp3`. To analyze the bass activity and restrict the data processed to 30 seconds, the BPM counter adds the following arguments to the previous command line:

```
... bandpass 100 1 trim 60 30
```

The bandpass filter removes frequencies outside the range of a bass drum by applying a 3dB damper per octave. The `trim` filter fast forwards 60 seconds into the track and extracts the data for the next 30 seconds when it gets there, so the script doesn't have to run through the entire song to find meaningful data.

## Disco Hoofing, and Heroic Tenor

Figure 7 shows the filtered audio files for various titles. The bass drum in "I Feel Fine" (upper right) and the booming synthetic drum in David Guetta's "Memories" (lower right) create clean peaks, which the script identifies and converts to beats per minute by some simple stretching. In the case of classical music, or acoustic guitar songs like "I Got a Name" by Jim Croce, the signal sits at a very low, more or less constant value, and the algorithm returns a value of `0`.

The bass signal values of fast punk songs like "Rich Lips" by Blink-182 vary dramatically and don't necessarily occur in regular intervals, but approximate acquisition of the peaks will typically give you meaningful BPM values.

After opening a connection to the database file, Listing 2 calls the `bpm_update()` function in line 43. The select query in line 48 returns the paths of all the tracks managed by Banshee as URIs of the format `file://path/file`. Because these URIs encode blanks as `%20`, the `uri_escape()` function from the CPAN URI::Escape module converts them back to normal blanks. Line 62 removes the leading `file://`, and, hey presto, you

Mike: Figure 7 says "No Getting Over." OK as set?? -rls

Mike: uri_unescape as in line 61?? -rls

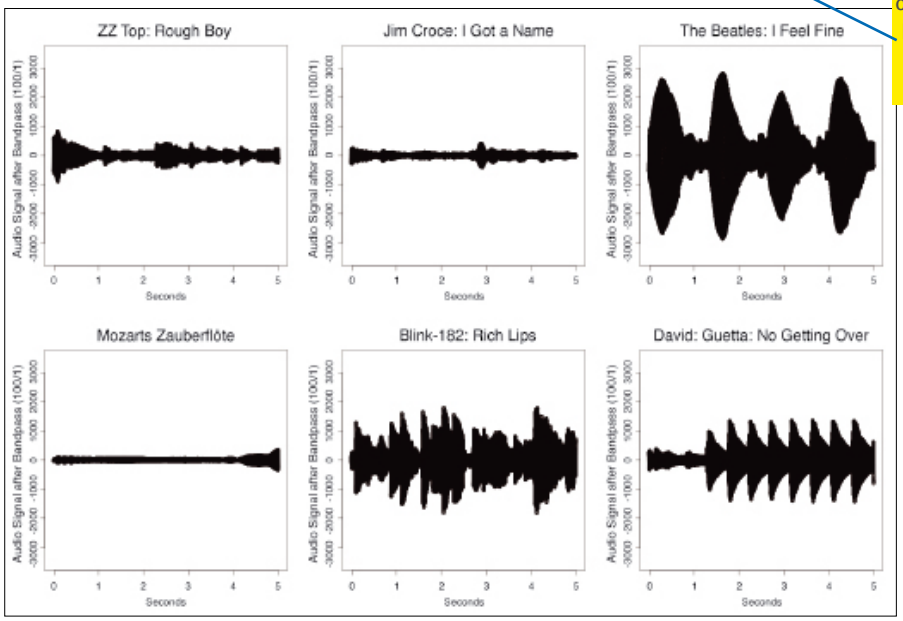



**Figure 7:** The bass lines of songs after applying a narrow bandpass filter.

have the Unix path to the audio file in `$file`.

The second SQL command, which line 52 prepares with matching placeholders, updates the value in the BPM column of the database, by using the URI as a selection criterion for `WHERE`. Line 64 uses `execute()` to send the update with the inserted URI and BPM parameters to the database.

While the `while` loop iterates across all the audio files, the SQL statement is stored in `$upd_sth`, and line 64 just needs to call it with new parameters each time. At the end of the `while` loop, `finish()` releases the internally created data structures.

## Reducing the Mass of Data

The `bpm()` function in line 72 handles the task of computing the BPM value for an audio file. If a `.raw` file has been passed in to it, line 79 picks up the file. However, in most cases, it will probably be a `.wav`, `.mp3`, `.ogg`, or something similar. The `sox` command in line 87, which is called using the CPAN module Sysadm::Install's `tap()` function, extracts 30 seconds of music after the one-minute marker, runs it through the narrow bandpass, and stores the resulting raw data in a `.raw` file of the same name.

To keep the mass of data somewhat tolerable, it reduces the sampling rate to the value set for `$SAMPLE_RATE` in line 16 – that is, to 10,000 per second.

The `samples()` function in line 111 then uses `sysread()` to parse the values stored in the `.raw` file in four-byte steps (two channels with two bytes each) and uses Perl's internal `unpack()` function with the `'ss'` placeholder in line 124 to extract the two signed integers. It ignores the value for the second channel in `$c2` (be-

### LISTING 2: banshee-bpm-update

```
001 #!/usr/local/bin/perl -w
002 ##############################
003 # banshee-bpm-update
004 # Mike Schilli, 2011
005 # (m@perlmeister.com)
006 ##############################
007 use strict;
008 use Log::Log4perl qw(:easy);
009 use DBI qw(:sql_types);
010 use DBD::SQLite;
011 use Sysadm::Install qw(tap);
012 use URI::Escape;
013 use File::Temp qw(tempfile);
014 use POSIX;
015
016 my $SAMPLE_RATE = 10_000;
017 my $OFFSET      = 60;
018 my $SAMPLE_SECS = 30;
019 my $MIN_SIZE    = 500;
020 my $MIN_DROP    = 0.7;
021 my $NWINDOWS    = 20;
022
023 Log::Log4perl->easy_init(
024  {
025   level    => $INFO,
026   category => "main"
027  }
028 );
029 my $db = glob "~/.config" .
030      "/banshee-1/banshee.db";
031 my $dbh = DBI->connect(
032  "dbi:SQLite:$db",
033  "", "",
034  {
035   RaiseError => 1,
036   AutoCommit => 1
037  }
038 );
039 bpm_update($dbh);
040 $dbh->disconnect();
041
042 ##############################
043 sub bpm_update {
044 ##############################
045  my ($dbh) = @_;
046
047  my $sth = $dbh->prepare(
048 "SELECT Uri FROM CoreTracks "
049      . "WHERE BPM = 0");
050  $sth->execute();
051
052  my $upd_sth = $dbh->prepare(
053     "UPDATE CoreTracks " .
054     "SET BPM=? " .
055     "WHERE Uri = ?");
056
057  while ((my $uri) =
058   $sth->fetchrow_array())
059  {
060   my $file =
061     uri_unescape($uri);
062   $file =~ s#^file://##;
063   INFO "Updating $uri";
064   $upd_sth->execute(
065    bpm($file), $uri);
066  }
067  $upd_sth->finish();
068  $sth->finish();
069 }
070
071 ##############################
072 sub bpm {
073 ##############################
074  my ($file) = @_;
075
076  my $rawfile;
077
078  if ($file =~ /\.raw$/) {
079   $rawfile = $file;
080  } else {
081   $rawfile = File::Temp->new(
082    SUFFIX => ".raw",
083    UNLINK => 1
084   );
085
086   my ($stdout, $stderr, $rc)
087     = tap "sox", $file, "-r",
088     $SAMPLE_RATE,
089     "-c", 2, "-b", 16, "-t",
090     "raw",
091     "-e", "signed", $rawfile,
092     "bandpass", 100, 1,
093     "trim", $OFFSET,
094     $SAMPLE_SECS;
095
096   if ($rc) {
097    LOGWARN
098      "sox $file: $stderr";
099    return -1;
100   }
101  }
102
103  return raw_bpm(
104   samples(
105    $rawfile->filename
106   )
107  );
108 }
109
110 ##############################
111 sub samples {
112 ##############################
113  my ($file) = @_;
114
115  my @vals = ();
116  sysopen FILE, "$file",
117    O_RDONLY
```

cause it is identical). It only sends the value for the first channel from `$c1` to the end of the results array `@vals` if it is above the threshold value of `$MIN_SIZE`. This value, which is set to `500` in line 19, is designed to prevent the maximum search from getting lost in minimal signals contained in quiet passages.

## Climbing Mountains

The script has no way of knowing in advance how many peak values it will find in the data array, and it has no idea of their size. Thus, it uses the "mountain climber method" (Figure 8), which means it investigates all the signal amplitudes in a short time window (e.g., 1/20th of a second) and stores the maximum value found within. If the local maximum in the next time window is larger than the stored value, the signal plot is ascending and the algorithm sets a flag. If the maximum value in the following time window in this mode is then smaller, the global maximum signal has just been passed and the script increments the counter.

This method works because the maximum number of maximum values to find is limited in the upward direction. BPM values greater than 600 don't make any sense and can be ignored. Thus, we will not discover two maximum values within two time windows of 1/20th second.

The `raw_bpm()` function picks up the data array for a channel and sets off to find the peak. Line 137 defines the time window width by dividing the number of data points by the required time window frequency (`$NWINDOWS`, set to 20 windows per second in line 21) times the number of sample seconds. The `$slope` variable marks the flag, which the algorithm uses to determine whether it is currently in an upward (`"up"`) or downward (`"down"`) trend. `raw_bpm()` stores the last global maximum in `$pmax`, and the local maximum of the window currently under investigation is in `$max`. To avoid minor signal fluctuation causing the method to trip over its own toes, `$MIN_DROP` uses a value of `0.7` to stipulate that the signal must fall by at least 30% to determine the previous peak as a maximum and increment the `$bumps` counter.

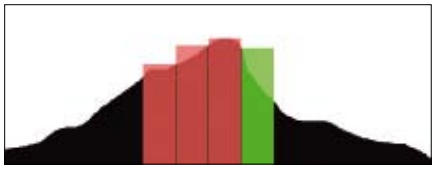


**Figure 8:** Mountain climber method.

Line 177 divides the number of maximum values found by the length of the data area under investigation in seconds and multiplies the results by `60.0` to achieve a BPM value for one minute.

## Installation

The Perl modules you additionally need to install for this project (DBI, DBD::SQLite, Sysadm::Install, URI::Escape, and Log::Log4perl) are available either as packages in your choice of distribution (e.g., libdbi-perl, libdbd-sqlite-perl, etc. for Ubuntu) or can be transferred to your local system using a CPAN shell.

The `sox` package is not universally available with MP3 support. If your local legislation prohibits the use of this format, you can easily work with WAV or OGG files instead.

If you have a large collection of music, you will probably want to convert Banshee to MySQL in the tools dialog for reasons of performance. Because DBI is database independent, you only need to modify the `connect()` line in the script (line 31). Instead of `"dbi:SQLite:$db"`, you should stipulate `"dbi:mysql:`*`dbname`*`"` and insert your username and password where the SQLite version has two blank strings. Another idea for automatic classification of audio files would be distinguishing between audio books, spoken word, and music. Or, you might want to analyze the harmonies to be able to listen to happy-sounding music on some days and leave minor chords or disharmony for others. ■■■

## INFO

[1] Banshee: *http://banshee.fm*
[2] Listings for this article: *http://www.linux-magazine.com/Resources/Article-Code*

## LISTING 2: banshee-bpm-update (continued)

```
118    or LOGDIE "$file: $!";
119
120  while (
121   sysread(FILE, my $val, 4))
122  {
123   my ($c1, $c2) =
124     unpack 'ss', $val;
125   $c1 = 0 if $c1 < $MIN_SIZE;
126   push @vals, $c1;
127  }
128  close FILE;
129  return @vals;
130 }
131
132 ############################
133 sub raw_bpm {
134 ############################
135  my (@samples) = @_;
136
137  my $win =
138    scalar @samples /
139    (
140   $NWINDOWS * $SAMPLE_SECS);
141  my ($bumps, $pmax, $slope) =
142    (0, 0, "up");
143
144  for (
145   my $o = 0 ;
146   $o <= $#samples - $win ;
147   $o += $win
148    )
149  {
150   my $max = 0;
151   for (
152    my $i = $o ;
153    $i <= $o + $win ;
154    $i++
155     )
156   {
157    if ($samples[$i] > $max) {
158     $max = $samples[$i];
159    }
160   }
161
162   if ($slope eq "up") {
163    if (
164     $max < $MIN_DROP * $pmax)
165    {
166     $slope = "down";
167     $bumps++;
168    }
169   } else {
170    $slope = "up"
171      if $max > $pmax;
172   }
173   $pmax = $max;
174  }
175
176  return
177    int($bumps /
178     $SAMPLE_SECS *
179     60.0)
180    || 1;
181 }
listing text here
```